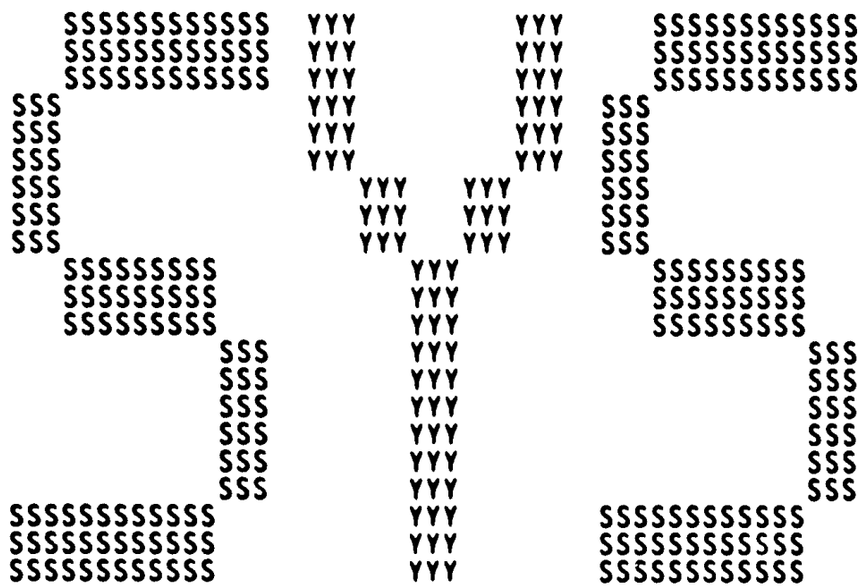

```
LL          NN      NN  MM      MM    SSSSSSSS  UU      UU  BBBBBBBB
LL          NN      NN  MM      MM    SSSSSSSS  UU      UU  BBBBBBBB
LL          NN      NN  MMMM  MMMM  SS          UU      UU  BB      BB
LL          NN      NN  MMMM  MMMM  SS          UU      UU  BB      BB
LL          NNNN    NN  MM  MM  MM  SS          UU      UU  BB      BB
LL          NNNN    NN  MM  MM  MM  SS          UU      UU  BB      BB
LL          NN  NN  NN  MM      MM    SSSSSS    UU      UU  BBBBBBBB
LL          NN  NN  NN  MM      MM    SSSSSS    UU      UU  BBBBBBBB
LL          NN    NNNN  MM      MM          SS  UU      UU  BB      BB
LL          NN    NNNN  MM      MM          SS  UU      UU  BB      BB
LL          NN      NN  MM      MM          SS  UU      UU  BB      BB      ....
LL          NN      NN  MM      MM          SS  UU      UU  BB      BB      ....
LLLLLLLLLL  NN      NN  MM      MM  SSSSSSSS    UUUUUUUUUU  BBBBBBBB        ....
LLLLLLLLLL  NN      NN  MM      MM  SSSSSSSS    UUUUUUUUUU  BBBBBBBB        ....

LL            IIIIII      SSSSSSSS
LL            IIIIII      SSSSSSSS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II        SSSSSS
LL              II        SSSSSS
LL              II              SS
LL              II              SS
LL              II              SS
LL              II              SS
LLLLLLLLLL    IIIIII    SSSSSSSS
LLLLLLLLLL    IIIIII    SSSSSSSS
```

Table of contents

(3)	643	DATA DEFINITIONS	
(6)	775	DELETE_ENTRY	- DELETE ONE LOGICAL NAME TABLE ENTRY
(7)	842	DELETE_LNMB	- DELETE LOGICAL NAME BLOCK
(8)	949	DELETE_NAMES	- SCAN HASH TABLE AND DELETE NAMES
(9)	1003	DELETE_TABLE	- DELETE A LOGICAL NAME TABLE
(10)	1068	LNM$CHECK_PROT	- CHECK ACCESS TO A LOGICAL NAME TABLE
(11)	1167	LNM$DELBLK	- DELETE P1 OR S0 PAGED BLOCK
(12)	1194	LNM$DELETE	- DELETE LOGICAL NAME TABLE ENTRY
(13)	1244	LNM$DELETE_LNMB	- DELETE LOGICAL NAME TABLE ENTRY PLUS ALIASES
(14)	1312	LNM$DELETE_HASH	- DELETE ALL ENTRIES IN A HASH TABLE
(15)	1352	LNM$DELETE_TAB	- DELETE ALL ENTRIES IN A LOGICAL NAME TABLE
(16)	1387	LNM$INIT_PROT	- INIT A LOGICAL NAME TABLE'S OBJECT RIGHTS BLOCK
(17)	1438	LNM$INSLOGTAB	- INSERT IN LOGICAL NAME TABLE BY ADDRESS
(18)	1755	LNM$SEARCHLOG	- SEARCH FOR LOGICAL NAME
(19)	1882	LNM$SEARCH_ONE	- SEARCH FOR LOGICAL NAME AND RETURN TRANSLATION
(20)	1976	LNM$FIRSTTAB	- SEARCH FOR FIRST TABLE NAME
(21)	2024	LNM$PRESEARCH	- FIND FIRST CANDIDATE NAME
(22)	2105	LNM$CONTSEARCH	- FIND NEXT CANDIDATE NAME
(23)	2199	LNM$HASH	- HASHING ALGORITHM
(24)	2235	LNM$LOOKUP	- LOOKUP TABLE NAME
(25)	2284	LNM$SETUP	- SETUP TO PROCESS TABLE NAME
(26)	2335	LNM$TABLE	- PROCESS LOGICAL NAME TABLE
(27)	2420	LNM$TABLE_SRCH	- PROCESS LOGICAL NAME TABLE
(28)	2516	LNM$TBL_CACHE	- SEARCH LOGICAL NAME TABLE TRANSLATION CACHE
(29)	2573	LNM$PROBER	- PROBE LOGICAL NAME DESCRIPTOR FOR READ ACCESS
(30)	2617	LNM$LOCKR	- LOCK LOGICAL NAME TABLE FOR READ ACCESS
(30)	2618	LNM$LOCKW	- LOCK LOGICAL NAME TABLE FOR WRITE ACCESS
(31)	2644	LNM$UNLOCK	- UNLOCK LOGICAL NAME TABLE

```
0000     1          .TITLE  LNMSUB - LOGICAL NAME RELATED SUBROUTINES
0000     2          .IDENT  'V04-000'
0000     3
0000     4  ;
0000     5  ;*****************************************************************************
0000     6  ;*                                                                           *
0000     7  ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
0000     8  ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0000     9  ;*  ALL RIGHTS RESERVED.                                                     *
0000    10  ;*                                                                           *
0000    11  ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED     *
0000    12  ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE    *
0000    13  ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER     *
0000    14  ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY     *
0000    15  ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY     *
0000    16  ;*  TRANSFERRED.                                                             *
0000    17  ;*                                                                           *
0000    18  ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE     *
0000    19  ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
0000    20  ;*  CORPORATION.                                                             *
0000    21  ;*                                                                           *
0000    22  ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS     *
0000    23  ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
0000    24  ;*                                                                           *
0000    25  ;*                                                                           *
0000    26  ;*****************************************************************************
0000    27  ;
0000    28  ; LOGICAL NAME RELATED SUBROUTINES ( REPLACES LOGNAMSUB MODULE )
0000    29  ;
0000    30  ; David W. Thiel    29-Oct-1982
0000    31  ;
0000    32  ; MODIFIED BY:
0000    33  ;
0000    34  ;       V03-031 RAS0330         Ron Schaefer            31-Jul-1984
0000    35  ;               Fix basic name lookup algorithm so that case blind
0000    36  ;               lookups can find lower case names.
0000    37  ;
0000    38  ;       V03-030 ACG0440         Andrew C. Goldstein,    24-Jul-1984  10:36
0000    39  ;               Add ref count field to ORB
0000    40  ;
0000    41  ;       V03-029 LMP0275         L. Mark Pilant,         12-Jul-1984  20:03
0000    42  ;               Initialize the ACL info in the ORB to be a null descriptor
0000    43  ;               list rather than an empty queue.  This avoids the overhead
0000    44  ;               of locking and unlocking the ACL mutex, only to find out
0000    45  ;               that the ACL was empty.
0000    46  ;
0000    47  ;       V03-028 RAS0319         Ron Schaefer                    29-Jun-1984
0000    48  ;               Add a simple logical name table name translation cache
0000    49  ;               to the recursive table name lookup algorithm in order to
0000    50  ;               (hopefully) improve the performance of logical name processing.
0000    51  ;               The methodology is as follows:
0000    52  ;
0000    53  ;               In per-process space, there exists a small queue of fixed-len
0000    54  ;               entries that contain a table name LNMB addr, a process and
0000    55  ;               system directory incarnation sequence number and upto
0000    56  ;               LNMC$K_NUM_ENTRIES worth of table header (LNMTH) addresses
0000    57  ;               that are the recursive tables identified by that table name.
```

```
0000     58 ;              When doing a table lookup;
0000     59 ;              We lookup the table name
0000     60 ;              normally.  We then check the cache for a matching LNMB.
0000     61 ;              If there is a cache entry and the directory incarnation numbers
0000     62 ;              are still valid, then we use the vector of LNMTH entries
0000     63 ;              rather than looking them up by name.
0000     64 ;              If no valid cache entry is found, we take the LRU cache
0000     65 ;              entry, initialize it for this name and use the normal
0000     66 ;              table lookup procedures, except that as a table header is
0000     67 ;              found, we store it's address in the cache entry for future
0000     68 ;              reference.
0000     69 ;              If we use up all cache entries before finding the correct
0000     70 ;              table, the cache entries will run out.  In that case, we
0000     71 ;              reset the cache index to start building a new cache entry
0000     72 ;              table from the beginning since it is impossible to relate
0000     73 ;              a given cache entry back to the corresponding name.
0000     74 ;              If the number of table entries, exceeds the cache table size
0000     75 ;              then we give up and do things the slow way.
0000     76 ;
0000     77 ;       V03-027 RAS0316         Ron Schaefer                    25-Jun-1984
0000     78 ;              Add new LNM$DELETE_HASH routine to delete all logical names
0000     79 ;              within a given hash table and having a given or outer
0000     80 ;              access mode.  This is ECO 4 in the FT2 update.
0000     81 ;
0000     82 ;       V03-026 RAS0312         Ron Schaefer                    18-Jun-1984
0000     83 ;              Prevent accvio in searching tables if no process logical
0000     84 ;              name table/hash table/directory has been created.
0000     85 ;              Fix alignment of ORB block in table.
0000     86 ;
0000     87 ;       V03-025 TMK0018         Todd M. Katz                    24-Apr-1984
0000     88 ;              Change the name of the logical name mutex from LOG$AL_MUTEX
0000     89 ;              to LNM$AL_MUTEX.
0000     90 ;
0000     91 ;       V03-024 TMK0017         Todd M. Katz                    21-Apr-1984
0000     92 ;              I have changed the interface to the routine LNM$DELETE_LNMB.
0000     93 ;              This routine is now called with the logical name block it is
0000     94 ;              to delete together with all of the outer access mode aliases
0000     95 ;              of the logical name represented by the logical name block.
0000     96 ;              Previously the interface to this routine consisted of it being
0000     97 ;              called with a logical name descriptor, containing table header
0000     98 ;              address, and the access mode of the innermost logical name
0000     99 ;              to be deleted, despite the fact that the calling routine always
0000    100 ;              had the logical name block for the innermost access mode logical
0000    101 ;              name to be deleted.
0000    102 ;
0000    103 ;              Make LNM$DELETE a local routine. This routine is only called by
0000    104 ;              the routines within this module.
0000    105 ;
0000    106 ;              The performance measurement cell used to monitor the rate of
0000    107 ;              logical name translations is currently located within the
0000    108 ;              internal logical name routine LNM$SEARCHLOG. Unfortunately,
0000    109 ;              because of its current placement, any attempts to delete
0000    110 ;              specific logical names will also increment this counter. This
0000    111 ;              is because the system service $DELLNM will call the routine
0000    112 ;              LNM$SEARCHLOG in such a situation. Therefore, in order to be
0000    113 ;              able to make a more accurate measurement of the overall rate
0000    114 ;              of logical name translations, I have decided to move this
```

```
0000    115 ;               performance measurement cell from its current single location
0000    116 ;               to several more appropriate locations. One of these new
0000    117 ;               locations is within the internal routine LNM$SEARCH_ONE just
0000    118 ;               before the call to LNM$SEARCHLOG.
0000    119 ;
0000    120 ;       V03-023 CWH3023         CW Hobbs                        14-Apr-1984
0000    121 ;               Fix a broken branch.
0000    122 ;
0000    123 ;       V03-022 TMK0016         Todd M. Katz                    11-Apr-1984
0000    124 ;               Make a change to DELETE_LNMB such that when this routine is
0000    125 ;               called to delete a shareable logical name table and its
0000    126 ;               associated Object Rights Block, it cleanups up the ORB first
0000    127 ;               before deleting the logical name block and ORB.
0000    128 ;
0000    129 ;       V03-021 TMK0015         Todd M. Katz                    10-Apr-1984
0000    130 ;               Change LNM$SEARCH_ONE to position past all translation blocks
0000    131 ;               with negative indexes (ie - those reserved for system use) to
0000    132 ;               the first translation block with a non-negative index or to the
0000    133 ;               last translation block, whichever comes first. If the
0000    134 ;               translation block positioned to is not the last one and has an
0000    135 ;               index of 0, then return success together with the contents of
0000    136 ;               the entire translation block in the user supplied buffer.
0000    137 ;               Otherwise, return an error of SS$_NOLOGNAM.
0000    138 ;
0000    139 ;       V03-020 TMK0014         Todd M. Katz                    02-Apr-1984
0000    140 ;               I have made the following optimizations and bug fixes to the
0000    141 ;               routines within this module.
0000    142 ;
0000    143 ;               1. I have changed how LNMBs are ordered within a hash bucket
0000    144 ;                  both to increase performance and to fix a problem in hash
0000    145 ;                  bucket searching. The new ordering is first by name string
0000    146 ;                  length, then my name string, then by containing table header
0000    147 ;                  address, and finally by access mode. Previously, the LNMBs
0000    148 ;                  were ordered by access mode before containing table header
0000    149 ;                  address.
0000    150 ;
0000    151 ;                  What this new ordering gains is the ability to use the
0000    152 ;                  containing table header address in determining when searches,
0000    153 ;                  such as those directed by LNM$SEARCHLOG, should terminate for
0000    154 ;                  a current containing table header address. With the old
0000    155 ;                  ordering, containing table header addresses could not be used
0000    156 ;                  to terminate a search. Even if the target LNMB was not found
0000    157 ;                  among the user mode LNMBs, the search would still have to
0000    158 ;                  continue with the supervisor, executive, and kernel mode
0000    159 ;                  LNMBs. Basically, this meant ignoring the ordering of LNMBs
0000    160 ;                  by containing table header addresses. Unfortunately, this
0000    161 ;                  "ignoring" was not being done, and this resulted in the
0000    162 ;                  inability to find certain logical names even though they were
0000    163 ;                  present. When I changed the ordering of LNMBs within the hash
0000    164 ;                  bucket, not only did I gain the ability to use the ordering
0000    165 ;                  of LNMBs by containing table header addresses to terminate a
0000    166 ;                  search, and thus increase performance, but at the same time
0000    167 ;                  I eliminated this particular problem which was causing
0000    168 ;                  certain $TRNLNMs to fail when they should have succeeded.
0000    169 ;
0000    170 ;               2. The routine LNM$CONTSEARCH makes the assumption that when
0000    171 ;                  it is directly called, NT_L_THREAD contains the address
```

```
0000   172 ;                of the previous LNMB block, and the search is to continue
0000   173 ;                with the LNMB which follows it. If there is no possibility
0000   174 ;                that the contents of the hash bucket could have changed since
0000   175 ;                the last search was done, then the name string of the first
0000   176 ;                LNMB this routine looks at, the one pointed at by the LNMB
0000   177 ;                whose address is contained within NL$L_THREAD, is guarenteed
0000   178 ;                to match the target name string. Therefore, there is no need
0000   179 ;                to make the string comparison which is normally done because
0000   180 ;                the outcome is already known, and in fact, an increase in
0000   181 ;                performance can be realized if the execution of this CMPC3
0000   182 ;                is eliminated.
0000   183 ;
0000   184 ;                What I have done is define a bit NT_V_MODIFY within the
0000   185 ;                NT_B_FLAGS field of the translation block. This bit is
0000   186 ;                set as part of the initialization of a name translation
0000   187 ;                block whenever the possibility exists that LNM$CONTSEARCH
0000   188 ;                maybe called after modifying the sequence of LNMBs residing
0000   189 ;                within a hash bucket. At the present two routines,
0000   190 ;                LNM$INSLOGTAB and LNM$DELETE_LNMB, set this bit as part of
0000   191 ;                the process of allocating and initializing their name
0000   192 ;                translation blocks. When LNM$CONTSEARCH notices that this bit
0000   193 ;                is clear it skips the initial CMPC3 assuming that the name
0000   194 ;                strings are equivalent. Only this initial CMPC3 can be
0000   195 ;                skipped - all remaining string comparisons are required and
0000   196 ;                are performed as is found to be necesary.
0000   197 ;
0000   198 ;            3.  I have also made numerous micro-optimizations to the routine
0000   199 ;                within this module.
0000   200 ;
0000   201 ;            4.  I have changed this module to use the symbol LNM$C_MAXDEPTH
0000   202 ;                to define the maximum logical name recursion depth instead
0000   203 ;                of the local symbol RT_C_MAXDEP.
0000   204 ;
0000   205 ;            5.  I have made two changes to LNM$SEARCH_ONE. Before calling
0000   206 ;                LNM$LOCKR to lock the logical name mutex for reading, this
0000   207 ;                routine saves the current IPL on the stack, and then restores
0000   208 ;                IPL to this value after unlocking the logical name mutex by
0000   209 ;                calling LNM$UNLOCK. This change is required because LNM$LOCKR
0000   210 ;                exits with IPL set to IPL$_AST, and therefore, IPL may have
0000   211 ;                to be set back to its inital value before LNM$SEARCH_ONE
0000   212 ;                exits. The reason why IPL is not lowered until after
0000   213 ;                LNM$UNLOCK returns is that the mutex locking and unlocking
0000   214 ;                routines make the assumption that IPL does not fall below
0000   215 ;                IPL$_AST while the current process has the mutex locked.
0000   216 ;
0000   217 ;                The second change I have made is a bug fix. LNM$SEARCH_ONE
0000   218 ;                was checking access to process-private logical name tables
0000   219 ;                when the logical name found was contained within one of them.
0000   220 ;                Such a check is un-necessary, and in fact must not be done.
0000   221 ;                Such tables do not have an Object Rights Block associated
0000   222 ;                with them, and LNM$CHECK_PROT will accvio if it is called
0000   223 ;                with a table header that does not have an associated ORB.
0000   224 ;
0000   225 ;            6.  I have fixed a day-one implementation problem in LNM$HASH
0000   226 ;                that was making the hashing routine sensitive to the
0000   227 ;                alignment of the address of the string this routine is
0000   228 ;                supposed to hash (and why this was ever working I'll never
```

```
0000   229 ;                know). What LNM$HASH should have been doing, and does now is:
0000   230 ;
0000   231 ;                1.) Save the number of bytes in the string.
0000   232 ;                2.) Divide the number of bytes in the string by 4.
0000   233 ;                3.) Process the string, four bytes at a time, to compute the
0000   234 ;                    hash code.
0000   235 ;                4.) Retrieve the string size, and use the low-order two bits,
0000   236 ;                    in effect the remainder from the integer division in 1.),
0000   237 ;                    to determine how many bytes of the string have not yet
0000   238 ;                    participated in the computation of the hash code, and to
0000   239 ;                    direct their participation.
0000   240 ;                5.) Complete computation of the hash code.
0000   241 ;
0000   242 ;                In step 1.), instead of saving the number of bytes in the
0000   243 ;                string, LNM$HASH was saving the string address. This made
0000   244 ;                the computation of the hash code sensitive to the alignment
0000   245 ;                of the string address passed to it. Inotherwards, if the
0000   246 ;                name string FOO was presented to LNM$HASH as being at address
0000   247 ;                500 at one time, and as being at address 601 a second time,
0000   248 ;                the two hash code values determined for FOO would be
0000   249 ;                different - an extremely serious problem. A given name must
0000   250 ;                always hash to a constant value.
0000   251 ;
0000   252 ;        V03-019 TMK0013         Todd M. Katz                    30-Mar-1984
0000   253 ;                Modify the logical name system services to make use of the
0000   254 ;                updated internal protection checking mechanisms. What this
0000   255 ;                requires is modification to the routine LNM$INIT_PROT, so that
0000   256 ;                it initializes a quad-word aligned Object Rights Block for
0000   257 ;                shareable logical name tables in place of an un-aligned CHIP
0000   258 ;                protection template, and a modification to the routine
0000   259 ;                LNM$CHECK_PROT, so that it makes use of the new internal
0000   260 ;                check protection system service interface.
0000   261 ;
0000   262 ;        V03-018 TMK0012         Todd M. Katz                    22-Mar-1984
0000   263 ;                Fix a bug in logical name table processing. When a translation
0000   264 ;                has the attribute TERMINAL, the translation string must be the
0000   265 ;                the name of a logical name table, and LNM$TABLE remembers this
0000   266 ;                for the next level of recursion by setting the bit RT_V_TERM
0000   267 ;                within the RT_B_FLAGS field of the table recursion control
0000   268 ;                block. If the translation string is the name of a logical name
0000   269 ;                table then LNM$TABLE makes a successful exit with this bit
0000   270 ;                still set within the recursion control block. Then if for
0000   271 ;                some reason LNM$TABLE is called once more to continue the
0000   272 ;                recursion and find the next table, because RT_V_TERM has not
0000   273 ;                been cleared, LNM$TABLE will immediately and incorrectly exit
0000   274 ;                with an error of SS$_IVLOGTAB. The fix to this problem is to
0000   275 ;                unconditionally clear this bit on entry to LNM$TABLE.
0000   276 ;
0000   277 ;        V03-017 TMK0011         Todd M. Katz                    21-Mar-1984
0000   278 ;                Make LNM$PRESEARCH, LNM$CONTSEARCH, LNM$SETUP, and LNM$TABLE
0000   279 ;                global, so that they maybe used by SHOW LOGICAL.
0000   280 ;
0000   281 ;        V03-016 TMK0010         Todd M. Katz                    07-Mar-1984
0000   282 ;                Logical name table name processing is recursive. A table
0000   283 ;                name is provided, and the corresponding logical name block is
0000   284 ;                located, if one exists, by hashing the name and looking for the
0000   285 ;                logical name block within the indicated hash buckets (first
```

```
0000   286 ;               searching the process-private name space and then searching the
0000   287 ;               system name space). If the logical name block exists, and it is
0000   288 ;               for a logical name table, then table processing terminates.
0000   289 ;               Otherwise, this procedure is repeated, in turn, for each of the
0000   290 ;               logical name's translations and for each of the translation's
0000   291 ;               translations, etc... until the first logical name table is
0000   292 ;               located, or until all possible translation paths have been
0000   293 ;               exhausted.
0000   294 ;
0000   295 ;               I have added an optimization to this recursive logical name
0000   296 ;               table name processing. This optimization consists of storing
0000   297 ;               the hash code value of each equivalence string within the
0000   298 ;               corresponding translation block. Because the very first step in
0000   299 ;               each recursive logical name block lookup is the hashing of the
0000   300 ;               target block's name string in order to provide the hash buckets
0000   301 ;               in which to concentrate the search, already having the
0000   302 ;               appropriate hash code means that this step of a logical name
0000   303 ;               block lookup maybe by-passed.
0000   304 ;
0000   305 ;               There are two exceptions to this optimization which will require
0000   306 ;               the target block's name to be hashed during a recursive logical
0000   307 ;               name block lookup. First, the target block's name will have to
0000   308 ;               be hashed during the very first recursive logical name block
0000   309 ;               lookup. This is because the hash value of this name string isn't
0000   310 ;               available. Secondly, because there is no way to distinquish
0000   311 ;               between a valid hash code of 0, and the total absence of a hash
0000   312 ;               code, this means that whenever one of these stored hash code
0000   313 ;               values is 0, the equivalence string within the corresponding
0000   314 ;               translation block will end up being hashed anyway during the
0000   315 ;               lookup of the target logical name block.
0000   316 ;
0000   317 ;               The modifications which are required in order to implement this
0000   318 ;               feature are as follows:
0000   319 ;
0000   320 ;               1. Increase the size of the fixed portion of each translation
0000   321 ;                  block by a word in both logical names and logical name
0000   322 ;                  tables. This word may potentially contain the translation
0000   323 ;                  string's hash code value.
0000   324 ;
0000   325 ;               2. Initialize this new field of each translation block with the
0000   326 ;                  hash code value of the corresponding equivalence string
0000   327 ;                  provided the translation block is part of a logical name
0000   328 ;                  (and not a logical name table) contained within a directory
0000   329 ;                  table. It is these names, and only these names, which are
0000   330 ;                  utilized in logical name table name processing, and thus,
0000   331 ;                  only these names have to have the hash code values of their
0000   332 ;                  equivalence strings computed and stored appropriately.
0000   333 ;
0000   334 ;                  The routine LNM$INSLOGTAB has been modified to perform this
0000   335 ;                  initialization whenever it determines that the logical name
0000   336 ;                  block it is inserting is for a logical name contained within
0000   337 ;                  the process or system directory logical name table.
0000   338 ;
0000   339 ;               3. The routine LNM$LOOKUP has been modified, so that it takes
0000   340 ;                  as additional input the hash code value of name string of the
0000   341 ;                  target logical name block this routine is to look up. This
0000   342 ;                  hash code value is placed into the hash function field of a
```

```
0000   343 ;                   name translation control block. This will result in the
0000   344 ;                   by-passing of the determination of the hash code value of the
0000   345 ;                   name string whenever the inputed hash code value is non-zero.
0000   346 ;
0000   347 ;               4. The routine LNM$TABLE has been modified, so that the hash
0000   348 ;                  code value of the name string of the next target logical name
0000   349 ;                  block is extracted from the translation block containing the
0000   350 ;                  name string. This hash code value, together with the name
0000   351 ;                  string, is then passed to LNM$LOOKUP forcing by-passing of
0000   352 ;                  the determination of the hash code value of the name string
0000   353 ;                  of the next target logical name block during its lookup.
0000   354 ;
0000   355 ;               5. Finally, a modification has been made to LNM$SETUP, the
0000   356 ;                  routine respcnsible for setting up and initialization the
0000   357 ;                  recursive logical name table name processing. Because the
0000   358 ;                  hash code of the name string of the initial target logical
0000   359 ;                  name block is not available, this routine must zero out the
0000   360 ;                  register in which LNM$LOOKUP expects to find the hash code
0000   361 ;                  of the logical name block it is to lookup. This will force
0000   362 ;                  the hash code value of this name string to be determined,
0000   363 ;                  which is proper since LNM$SETUP doesn't have it anyway.
0000   364 ;
0000   365 ;       V03-015 RAS0255         Ron Schaefer                    17-Feb-1984
0000   366 ;               Make LNM$M_CASE_BLIND work for DEC multinational characters.
0000   367 ;
0000   368 ;       V03-014 TMK0009         Todd M. Katz                    03-Feb-1984
0000   369 ;               If LNM$FIRSTTAB is unable to find an existing logical name
0000   370 ;               table within the input list of logical name tables names,
0000   371 ;               return an error of SS$_NOLOGTAB instead of an error of
0000   372 ;               SS$_NOLOGNAM.
0000   373 ;
0000   374 ;       V03-013 TMK0008         Todd M. Katz                    29-Dec-1983
0000   375 ;               Add the global routine LNM$DELETE_LNMB which takes as input
0000   376 ;               the address of a table header, a descriptor of the name of a
0000   377 ;               logical name table entry, and an access mode and deletes all
0000   378 ;               instances of the logical name table entry within the specified
0000   379 ;               logical name table at all access modes outer and equal to that
0000   380 ;               of the specified access mode.
0000   381 ;
0000   382 ;               Add the global routine LNM$SEARCH_ONE which takes as input:
0000   383 ;
0000   384 ;               1. A descriptor of a logical name table name.
0000   385 ;               2. A descriptor of a logical name.
0000   386 ;               3. A descriptor of an output buffer.
0000   387 ;               4. The PCB address.
0000   388 ;               5. An access mode.
0000   389 ;
0000   390 ;               This routine simulates a $TRNLNM returning a copy of the LNMX
0000   391 ;               translation block for translation index 0 in the specified
0000   392 ;               output buffer provided a logical name is found, and the found
0000   393 ;               logical name has a translation with an index of 0. This routine
0000   394 ;               does NOT do any type of arguement verification.
0000   395 ;
0000   396 ;               Make a small modification to DELETE_NAMES and LNM$DELETE
0000   397 ;               so that the directories, process and system, can never be
0000   398 ;               either explicitely or implicitely deleted. Also, return an
0000   399 ;               error from within LNM$INSLOGTAB if the caller attempted to
```

```
0000   400 ;                create a logical name table entry within one of the directories
0000   401 ;                with the same name and access mode as the directory itself.
0000   402 ;                This is done by never allowing a LNMB which has the
0000   403 ;                LNMB$V_NODELETE bit set within its LNMB$B_FLAGS field from being
0000   404 ;                deleted, and as the directory tables are the only tables that
0000   405 ;                are created with this attribute, this protects the directory
0000   406 ;                tables, and only the directory tables, from being deleted.
0000   407 ;
0000   408 ;                Make the global routines LNM$PRESEARCH, LNM$CONTSEARCH,
0000   409 ;                LNM$LOOKUP, LNM$SETUP, AND LNM$TABLE local routines.
0000   410 ;
0000   411 ;        V03-012 TMK0007         Todd M. Katz                    27-Dec-1983
0000   412 ;                LNM$INIT_PROT was clearing the member field of the CHIP template
0000   413 ;                UIC if the logical name table whose CHIP template was being
0000   414 ;                inited was a group table. This is no longer necessary because
0000   415 ;                group tables are now handcrafted, and can not be created by
0000   416 ;                means of the $CRELNT system service.
0000   417 ;
0000   418 ;        V03-011 TMK0006         Todd M. Katz                    14-Dec-1983
0000   419 ;                Make a change to recursive table lookup processing implemented
0000   420 ;                within the routine LNM$TABLE. Currently when the lookup of a
0000   421 ;                name fails it also terminates the recursive search for a logical
0000   422 ;                name table. The change is to not have a name lookup failure
0000   423 ;                automatically terminate a table lookup. Instead when the lookup
0000   424 ;                of a translation string fails, processing continues with the
0000   425 ;                next translation and an attempt is made to lookup its string.
0000   426 ;                Conceptually what this means is that non-existant logical name
0000   427 ;                tables in a list of logical name table names are "skipped over"
0000   428 ;                during the search for valid logical name tables. Currently, what
0000   429 ;                would happen is that such a search for valid tables will
0000   430 ;                immediately terminate with an error when the first non-existant
0000   431 ;                table was encountered in the list of logical name table names.
0000   432 ;
0000   433 ;                Also, change some PUSHRs into PUSHLs (or MOVQs) and POPRs into
0000   434 ;                POPLs (or MOVQs) where appropriate for performance reasons.
0000   435 ;
0000   436 ;        V03-010 TMK0005         Todd M. Katz                    26-Oct-1983
0000   437 ;                Quota handling when logical name tables are being created or
0000   438 ;                deleted is presently incorrect. Currently, when a logical name
0000   439 ;                table is created, quota consisting of the size of the new
0000   440 ;                logical name table plus any quota explicitely allocated to the
0000   441 ;                new table is subtracted from the quota holder of the parent
0000   442 ;                logical name. When a logical name table is deleted, the reverse
0000   443 ;                of this quota deduction takes place. This deduction scheme is
0000   444 ;                consistant but incorrect, and I have changed it by modifying
0000   445 ;                DELETE_LNMB and LNM$INSLOGTAB as follows:
0000   446 ;
0000   447 ;                When a logical name table is created, any quota explictely
0000   448 ;                allocated to the new table is deducted from the parent table's
0000   449 ;                quota holder as was previously being done; however, the size
0000   450 ;                of the new table itself is deducted from the quota holder of the
0000   451 ;                table that contains it (either the system or process directory
0000   452 ;                table). This is consistent with how logical names are handled,
0000   453 ;                and the philosophy that logical name tables are just logical
0000   454 ;                names with a special translation. When a logical name table is
0000   455 ;                deleted, the reverse of these two deductions takes place.
0000   456 ;
```

```
0000   457 ;               I have also removed the code for LNM$INSLOGN, LNM$TRANSLOGNAME,
0000   458 ;               and LNM$TRANSLATE. These routines are never called, and have
0000   459 ;               been commented out for months.
0000   460 ;
0000   461 ;       V03-009 ACG0354         Andrew C. Goldstein,    12-Sep-1983  21:55
0000   462 ;               Change RWACCESS field in CHIP$ block to FLAGS field
0000   463 ;
0000   464 ;       V03-008 TMK0004         Todd M. Katz                    31-Aug-1983
0000   465 ;               At the present time logical name blocks in each hash bucket are
0000   466 ;               ordered first by logical name size, then by access mode, then
0000   467 ;               alphabetically, by logical name, and finally by containing
0000   468 ;               table. The ordering of logical name blocks by access mode
0000   469 ;               before alphabetically meant that a supervisor mode "d" logical
0000   470 ;               name block would be found in the same hash bucket before an
0000   471 ;               executive mode "c" logical name block, and the explicit search
0000   472 ;               for "c" with the starting access mode specified as user or
0000   473 ;               supervisor would stop when the "d" logical name block was
0000   474 ;               encountered before the "c" was seen. Thus, "c" would never be
0000   475 ;               located unless an explicit search for an executive mode "c"
0000   476 ;               was initiated. This represented an error and is basically due
0000   477 ;               to the fact that searches for an explicit logical name are
0000   478 ;               performed for names at the given and inner access modes, while
0000   479 ;               the search process itself will stop as soon as a logical name
0000   480 ;               is encountered that collates higher than the logical name being
0000   481 ;               searched for, regardless if the name being searched for exists
0000   482 ;               at an inner access mode then the logical name block that
0000   483 ;               terminates the search. To fix this problem I have changed how
0000   484 ;               logical name blocks are ordered within a hash bucket. Logical
0000   485 ;               name blocks are now ordered first by the size of the logical
0000   486 ;               name, then alphabetically by logical name, then by access mode
0000   487 ;               of the logical name, and finally by the containing table's
0000   488 ;               table header address.
0000   489 ;
0000   490 ;       V03-007 TMK0003         Todd M. Katz                    09-Aug-1983
0000   491 ;               If LNM$INIT_PROT has been called to initialize the CHIP
0000   492 ;               protection template for a group logical name table (as
0000   493 ;               signalled by the setting of the bit LNMTH$V_GROUP), then zero
0000   494 ;               the member portion of the owner UIC field of the CHIP template
0000   495 ;               so that the group table does not have an owner.
0000   496 ;
0000   497 ;               Modify the routine LNM$CHECK_PROT so that it will grant access
0000   498 ;               to a logical name table, even when the caller would otherwise be
0000   499 ;               refused access based upon SOGW access protection, in two special
0000   500 ;               circumstances. First, if the logical name table is a group
0000   501 ;               logical name table (LNMTH$V_GROUP is set), the caller has the
0000   502 ;               GRPNAM privilege, the group logical name table is the caller's
0000   503 ;               group logical name table, and the caller has requested R or W
0000   504 ;               access to the table only, then return success granting access
0000   505 ;               to the table. Second if the logical name table is the system
0000   506 ;               logical name table (LNMTH$V_SYSTEM is set), the caller has the
0000   507 ;               SYSNAM privilege, and the caller has requested R or W access to
0000   508 ;               the table only, then return success granting access to the
0000   509 ;               table. These changes are required for compatibility reasons so
0000   510 ;               that access to the system and group tables is governed by the
0000   511 ;               same privileges across releases.
0000   512 ;
0000   513 ;               Comment out the unused routine LNM$INSLOGN.
```

```
0000   514 ;
0000   515 ;       V03-006 RAS0165         Ron Schaefer            5-Jul-1983
0000   516 ;               Correct RAS0158 and RAS0160 to ignore the caller's
0000   517 ;               access mode in the CHIP protection checking since the
0000   518 ;               logical name code uses a non-standard interpretation of
0000   519 ;               access mode.
0000   520 ;
0000   521 ;       V03-005 LMP0125         L. Mark Pilant,         26-Jun-1983  21:54
0000   522 ;               Change all referneces of CHIP$B_ACCESSOR_MODE to be
0000   523 ;               CHIP$L_ACCESSOR_MODE.
0000   524 ;
0000   525 ;       V03-004 RAS0160         Ron Schaefer                    16-Jun-1983
0000   526 ;               Add access mode of table to CHIP block and move
0000   527 ;               performance measurement cell to LNM$SEARCHLOG.
0000   528 ;
0000   529 ;       V03-003 RAS0158         Ron Schaefer                    25-May-1983
0000   530 ;               Add protection checking support subroutines LNM$INIT_PROT
0000   531 ;               to initialize the CHIP block for a table and LNM$CHECK_PROT
0000   532 ;               to check references to a table.
0000   533 ;
0000   534 ;       V03-002 TMK0002         Todd M. Katz                    25-Apr-1983
0000   535 ;               Fix several more bugs in these subroutines. The most
0000   536 ;               significant fix is one to LNM$SEARCHLOG. As this routine
0000   537 ;               locates the logical name tables in the list of such tables,
0000   538 ;               it searches for the presence of the given logical name within
0000   539 ;               them until a match is found. If the current logical name table
0000   540 ;               in the list of tables is shareable, only the shareable name
0000   541 ;               space is looked at for a match; likewise, if the current logical
0000   542 ;               name table is process-private, only the process-private name
0000   543 ;               space is referenced. The code that was making the distinction
0000   544 ;               between these two cases was incorrect, the result being that
0000   545 ;               the process-private name space was always being searched.
0000   546 ;               Furthermore, the code was incorrect in that processing of the
0000   547 ;               current table should be skipped if there is no name in the
0000   548 ;               same name space as the table itself. This was not being done
0000   549 ;               and allowed the possibility for access violations to occur.
0000   550 ;
0000   551 ;               In addition, fix the quota check in LNM$INSLOGN. The branch
0000   552 ;               following this check was signed, and it should have been
0000   553 ;               unsigned.
0000   554 ;
0000   555 ;               Also, increase the maximum recursion depth from 8 to 10.
0000   556 ;
0000   557 ;       V03-001 TMK0001         Todd M. Katz                    25-Mar-1983
0000   558 ;               Fix several bugs in these subroutines:
0000   559 ;
0000   560 ;               1. The PCB is only required on calls to LNM$TRNSLOGNAME,
0000   561 ;                  LNM$TRANSLATE, LNM$LOCKR, LNM$LOCKW, and LNM$UNLOCK.
0000   562 ;                  Therefore, remove the requirement that it be present in R4
0000   563 ;                  from all routines except for the above mentioned five.
0000   564 ;
0000   565 ;               2. When allocating and filling in the Recursive Table Name
0000   566 ;                  Control Block within LNM$FIRSTTAB and within LNM$SEARCHLOG,
0000   567 ;                  use a MOVZWL instead of a MOVZBL to fill in the access mode
0000   568 ;                  and set the case control bit because the latter occupies
0000   569 ;                  the first bit of the second byte of the source operand.
0000   570 ;
```

```
0000    571 ;               3. The table header of the process or system directory table is
0000    572 ;                  supposed to go in the TABID Field of the Name Translation
0000    573 ;                  Block when a table name is to be looked up by calling
0000    574 ;                  LNM$LOOKUP. This routine was placing t'e address of the table
0000    575 ;                  instead of the address of the table header in this field.
0000    576 ;
0000    577 ;               4. I have restructured the routine LNM$TABLE as follows: This
0000    578 ;                  routine now saves the address of the translation block it is
0000    579 ;                  working on in the Recursive Table Control Block before it
0000    580 ;                  looks up the translation string in the hash tables. The
0000    581 ;                  one exception is the original table name which is looked
0000    582 ;                  up without being saved. This allows continuation down the
0000    583 ;                  translation blocks of eight logical name blocks until a
0000    584 ;                  table is found as long as the names being looked up are in
0000    585 ;                  fact found. Previously, this information saving was being
0000    586 ;                  done incorrectly, and at the wrong time. This meant that
0000    587 ;                  the routine could not pick up and continue with the next
0000    588 ;                  translation block after the current translation thread
0000    589 ;                  dead ended.
0000    590 ;
0000    591 ;               5. I have restructured the routine LNM$CONTSEARCH as follows.
0000    592 ;                  As this routine is processing the LNMB Blocks within the
0000    593 ;                  current hash bucket it keeps the address of the previous
0000    594 ;                  LNMB within the THREAD cell of the Name Translation Control
0000    595 ;                  Block while the address of the LNMB it is working on is
0000    596 ;                  always in R1. Then whenever this routine exits, regardless
0000    597 ;                  of the status it exits with, R1 will always contain the
0000    598 ;                  address of the LNMB Block that stopped the search (or 0 if
0000    599 ;                  the list was exhausted) and NT_L_THREAD contains the address
0000    600 ;                  of the previous LNMB block. This provided sufficient
0000    601 ;                  information to allow new LNMB blocks to be inserted in the
0000    602 ;                  correct place regardless of where that place is.
0000    603 ;
0000    604 ;                  In addition this routine was performing a CMPC3, and then
0000    605 ;                  checking the state of a bit within a field pointed at by R3
0000    606 ;                  to determine whether the comparison is to be made case
0000    607 ;                  sensitive or blind. Of course, the CMPC3 changes R3, so this
0000    608 ;                  check must be made using the original contents of R3 which
0000    609 ;                  have been stored on the stack.
0000    610 ;
0000    611 ;               6. I have re-written LNM$INSLOGTAB. There were many problems
0000    612 ;                  with the old routine including the inability to correctly
0000    613 ;                  find the table header of the new table entry, an inability
0000    614 ;                  to just map a new entry to an existing entry when CREATE_IF
0000    615 ;                  had been specified, and the way in which new entries were
0000    616 ;                  linked into the exisiting list of entries within a hash
0000    617 ;                  hash bucket was completely incorrect.
0000    618 ;
0000    619 ;               7. I have have re-written the routine DELLNMB as DELETE_LNMB,
0000    620 ;                  the routine DELTAB as DELETE_NAMES, the routine LNM$DELETE,
0000    621 ;                  and the routine LNM$DELTAB as LNM$DELETE_TAB. I have changed
0000    622 ;                  the name of DEL1 to DELETE_ENTRY, added the routine
0000    623 ;                  DELETE_TABLE, and eliminated the routine LNM$DELHASH.
0000    624 ;
0000    625 ;               9. The status SS$_EXLNMQUOTA is returned instead of SS$_EXQUOTA.
0000    626 ;
0000    627 ;              10. Put the names of the routines on the subtitles.
```

```
0000    628 ;
0000    629 ;               11. The routine LNM$SEARCHLOG was incorrectly searching for a
0000    630 ;                   logical name. This routine first ascertains that the logical
0000    631 ;                   name exists at all, and then searches among the possible
0000    632 ;                   logical name tables one-by-one until it finds the first
0000    633 ;                   table that contains such a logical name. The first time this
0000    634 ;                   routine encountered a process-private table, it would
0000    635 ;                   return success even if the table did not contain the logical
0000    636 ;                   name. This was due to a slight mistake in the ordering of
0000    637 ;                   some of the instructions which has been corrected.
0000    638 ;
0000    639 ;               12. Re-write LNM$INSLOGN.
0000    640 ;--
```

```
0000    642
0000    643          .SBTTL  DATA DEFINITIONS
0000    644
0000    645 ;
0000    646 ; MACRO LIBRARY CALLS:
0000    647 ;
0000    648
0000    649          $ARMDEF                                 ;DEFINE ACCESS RIGHTS MASK
0000    650          $CADEF                                  ;DEFINE CONDITIONAL ASSEMBLY SWITCHES
0000    651          $CHPCTLDEF                              ;DEFINE CHIP PROTECTION CONTROL BLOCK OFFSET
0000    652          $DYNDEF                                 ;DEFINE STRUCTURE TYPE AND SUBTYPE CODES
0000    653          $LNMDEF                                 ;DEFINE LOGICAL NAME ATTRIBUTES
0000    654          $LNMSTRDEF                              ;DEFINE LOGICAL NAME BLOCKS OFFSETS
0000    655          $ORBDEF                                 ;DEFINE ORBJECT RIGHTS BLOCK OFFSETS
0000    656          $PCBDEF                                 ;DEFINE PCB OFFSETS
0000    657          $PRVDEF                                 ;DEFINE PRIVILEGE MASK OFFSETS
0000    658          $PSLDEF                                 ;DEFINE PROCESSOR STATUS FIELDS
0000    659          $SSDEF                                  ;DEFINE SYSTEM STATUS VALUES
0000    660
0000    661 ;
0000    662 ; ASSUMPTIONS ABOUT THE STRUCTURE AN OBJECT RIGHTS BLOCK AND A CHPCTL:
0000    663 ;
0000    664
0000    665          ASSUME  ORB$L_OWNER,          EQ, 0
0000    666          ASSUME  ORB$L_OWNER+4,        EQ, ORB$L_ACL_MUTEX
0000    667          ASSUME  ORB$L_ACL_MUTEX+4,    EQ, ORB$W_SIZE
0000    668          ASSUME  ORB$W_SIZE+2,         EQ, ORB$B_TYPE
0000    669          ASSUME  ORB$B_TYPE+1,         EQ, ORB$B_FLAGS
0000    670          ASSUME  ORB$B_FLAGS+3,        EQ, ORB$W_REFCOUNT
0000    671          ASSUME  ORB$W_REFCOUNT+2,     EQ, ORB$Q_MODE_PROT
0000    672          ASSUME  ORB$Q_MODE_PROT+8,    EQ, ORB$L_SYS_PROT
0000    673          ASSUME  ORB$L_SYS_PROT+4,     EQ, ORB$L_OWN_PROT
0000    674          ASSUME  ORB$L_OWN_PROT+4,     EQ, ORB$L_GRP_PROT
0000    675          ASSUME  ORB$L_GRP_PROT+4,     EQ, ORB$L_WOR_PROT
0000    676          ASSUME  ORB$L_WOR_PROT+4,     EQ, ORB$L_ACL_COUNT
0000    677          ASSUME  ORB$L_ACL_COUNT+4,    EQ, ORB$L_ACL_DESC
0000    678          ASSUME  ORB$L_ACL_DESC+4,     EQ, ORB$R_MIN_CLASS
0000    679          ASSUME  ORB$R_MIN_CLASS+ORB$S_MIN_CLASS,-
0000    680                                        EQ, ORB$R_MAX_CLASS
0000    681          ASSUME  ORB$R_MAX_CLASS+ORB$S_MAX_CLASS,-
0000    682                                        EQ, ORB$K_LENGTH
0000    683
0000    684          ASSUME  CHPCTL$L_ACCESS,      EQ, 0
0000    685          ASSUME  CHPCTL$L_ACCESS+4,    EQ, CHPCTL$L_FLAGS
0000    686          ASSUME  CHPCTL$L_FLAGS+4,     EQ, CHPCTL$B_MODE
0000    687          ASSUME  CHPCTL$B_MODE+4,      EQ, CHPCTL$C_LENGTH
0000    688 ;        .PAGE
```

```
                     690 ;**********************************************************************
                     691 ;
                     692 ; THIS CONTROL BLOCK IS ALSO LOCALLY DEFINED IN THE SHOW LOGICAL UTILITY
                     693 ; ANY CHANGES TO THIS DATA STRUCTURE MUST BE REFLECTED IN SHOW LOGICAL AS WELL.
                     694 ;
                     695 ;**********************************************************************
                     696
                     697 ;
                     698 ; BLOCK TO CONTROL RECURSIVE TABLE NAME TRANSLATION
                     699 ;
                     700 ;        31           25 24           16            9 8 7            0
                     701 ;       +---------------+---------------+-------------+---------------+
                     702 ; R5 -> | TRIES         | DEPTH         |           T C | ACCESS MODE   |
                     703 ;       +---------------+---------------+-------------+---------------+
                     704 ;       |               ADDRESS OF NAME BLOCK                           |
                     705 ;       +---------------+---------------+-------------+---------------+
                     706 ;       |                                                               |
                     707 ;       |       LNM$C_MAXDEPTH LONGWORDS TO SERVE AS STACK              |
                     708 ;       |                                                               |
                     709 ;       |                                                               |
                     710 ;       +---------------+---------------+-------------+---------------+
                     711 ;
                     712 ;       F IS SET TO INHIBIT FURTHER RECURSION
                     713 ;
00000000 0000       714 RT_W_R5=        0
00000000 0000       715 RT_B_ACMODE=    0
00000001 0000       716 RT_B_FLAGS=     1
00000008 0000       717 RT_V_CASE=      8                       ;CASELESS FLAG
00000001 0000       718 RT_M_CASE=      ^X1                     ;CASELESS MASK
00000009 0000       719 RT_V_TERM=      9                       ;INHIBIT RECURSION FLAG
00000002 0000       720 RT_M_TERM=      ^X2                     ;INHIBIT RECURSION MASK
         0000       721 ; BIT NUMBER 10 IS RESERVED.
         0000       722
00000002 0000       723 RT_B_DEPTH=     2                       ;RECURSION DEPTH
00000003 0000       724 RT_B_TRIES=     3                       ;RECURSION TRIES
000000FF 0000       725 RT_C_MAXTRIES=  255                     ;MAXIMUM NUMBER OF TRIES
00000004 0000       726 RT_L_CACHEPTR=  4                       ;POINTER TO CACHE ENTRY
00000008 0000       727 RT_L_STACK=     8
00000030 0000       728 RT_K_LENGTH=    RT_L_STACK+<4*LNM$C_MAXDEPTH>   ;BLOCK LENGTH
         0000       729
         0000       730 ;       .PAGE
```

```
                     0000   732 ;*****************************************************************************
                     0000   733 ;
                     0000   734 ; THIS CONTROL BLOCK IS ALSO LOCALLY DEFINED IN THE SHOW LOGICAL UTILITY
                     0000   735 ; ANY CHANGES TO THIS DATA STRUCTURE MUST BE REFLECTED IN SHOW LOGICAL AS WELL.
                     0000   736 ;
                     0000   737 ;*****************************************************************************
                     0000   738
                     0000   739 ;
                     0000   740 ; BLOCK TO CONTROL NAME TRANSLATION
                     0000   741 ;
                     0000   742 ;          31            25 24            16 15        A   8 7             0
                     0000   743 ;         +---------------+----------------+------------+---------------+
                     0000   744 ; R3 ->    HASH FUNCTION VALUE OR 0          |          M   C  ACCESS MODE   |
                     0000   745 ;         +---------------+----------------+------------+---------------+
                     0000   746 ;         |                          LENGTH OF NAME                      |
                     0000   747 ;         +---------------+----------------+------------+---------------+
                     0000   748 ;         |                          ADDRESS OF NAME                     |
                     0000   749 ;         +---------------+----------------+------------+---------------+
                     0000   750 ;         |                       TABLE HEADER ADDRESS OR 0              |
                     0000   751 ;         +---------------+----------------+------------+---------------+
                     0000   752 ;         |                       NAME BLOCK ADDRESS OR 0                |
                     0000   753 ;         +---------------+----------------+------------+---------------+
                     0000   754 ;
                     0000   755 ;
                     0000   756
00000000             0000   757 NT_W_R5=        0
00000000             0000   758 NT_B_ACMODE=    0
00000001             0000   759 NT_B_FLAGS=     1
00000001             0000   760 NT_M_CASE=      1                       ;CASELESS MASK
00000008             0000   761 NT_V_CASE=      8                       ;CASELESS FLAG
                     0000   762 ; BIT NUMBER 9 IS RESERVED.
00000004             0000   763 NT_M_MODIFY=    4                       ;HASH BUCKET SUSCEPTIBLE TO CHANGE MASK
0000000A             0000   764 NT_V_MODIFY=    10                      ;HASH BUCKET SUSCEPTIBLE TO CHANGE FLAG
                     0000   765
00000002             0000   766 NT_W_HASH=      2                       ;HASH FUNCTION OR 0
00000004             0000   767 NT_L_NAMLEN=    4                       ;LENGTH OF NAME
00000008             0000   768 NT_L_NAMADR=    8                       ;ADDRESS OF NAME
0000000C             0000   769 NT_L_TABID=     12                      ;TABLE HEADER ADDRESS OR 0
00000010             0000   770 NT_L_THREAD=    16                      ;LNMB POINTER
00000014             0000   771 NT_K_LENGTH=    20                      ;BLOCK LENGTH
                     0000   772
                     0000   773 ;       .PAGE
```

```
                     0000   775          .SBTTL  DELETE_ENTRY    - DELETE ONE LOGICAL NAME TABLE ENTRY
                     0000   776 ;+
                     0000   777 ; DELETE_ENTRY - DELETE ONE LOGICAL NAME TABLE ENTRY
                     0000   778 ;
                     0000   779 ; THIS ROUTINE IS CALLED TO DELETE ONE LOGICAL NAME TABLE ENTRY.
                     0000   780 ; IF THE ENTRY IS A TABLE HEADER, IT IS LINKED TO R5.
                     0000   781 ; OTHERWISE, THE LOGICAL NAME BLOCK IS DELETED.
                     0000   782 ; IN EITHER CASE, ALL SPECIAL INFORMATION (OTHER THAN THE TABLE
                     0000   783 ; HEADER) IS HANDLED.
                     0000   784 ;
                     0000   785 ; INPUTS:
                     0000   786 ;
                     0000   787 ;       R1 = ADDRESS OF ENTRY TO BE DELETED.
                     0000   788 ;       R5 = ADDRESS OF LIST OF LOGICAL NAME BLOCKS CONTAINING
                     0000   789 ;            TABLE HEADERS
                     0000   790 ;
                     0000   791 ;       IT IS ASSUMED THAT THE LOGICAL NAME MUTEX IS LOCKED FOR WRITE ACCESS,
                     0000   792 ;       AND THAT THE CALLER HAS THE PRIVILEGE OF DELETING THE LOGICAL NAME
                     0000   793 ;       TABLE ENTRY.
                     0000   794 ;
                     0000   795 ; OUTPUTS:
                     0000   796 ;
                     0000   797 ;       THE ENTRY IS REMOVED FROM ITS RESPECTIVE LOGICAL NAME TABLE AND THE
                     0000   798 ;       STORAGE IS RETURNED TO THE APPROPRIATE ALLOCATION REGION.
                     0000   799 ;
                     0000   800 ;       ADDITIONAL LOGICAL NAME BLOCKS CONTAINING TABLE HEADERS MAY BE
                     0000   801 ;       LINKED TO R5.
                     0000   802 ;       R0, R1, R2, AND R3 ARE MODIFIED.
                     0000   803 ;-
                     0000   804
                00000000    805          .PSECT  YF$$LNM
                     0000   806 DELETE_ENTRY:                                ;DELETE LOGICAL NAME TABLE ENTRY
     52   61    7D  0000    807          MOVQ    LNMB$L_FLINK(R1),R2         ;PICK BOTH LINKS
     63   82    DE  0003    808          MOVAL   (R2)+,LNMB$L_FLINK(R3)      ;STORE NEXT PTR IN PREVIOUS BLOCK
          03    13  0006    809          BEQL    5$                          ;THIS IS THE END OF THE LINE
     62   53    D0  0008    810          MOVL    R3,LNMB$L_BLINK-4(R2)       ;STORE PREVIOUS PTR IN NEXT BLOCK
  53    11    A1  9E  000B    811 5$:      MOVAB   LNMB$T_NAME(R1),R3          ;POINT TO COUNTED NAME STRING
     50   83    9A  000F    812 10$:     MOVZBL  (R3)+,R0                    ;LENGTH OF NAME
     53   50    C0  0012    813          ADDL2   R0,R3                       ;ADDRESS OF TRANSLATION
  05 63   02    E0  0015    814          BBS     #LNMX$V_XEND, -             ;END OF TRANSLATIONS
                     0019    815                  LNMX$B_FLAGS(R3),20$
17 01 A3   07    E0  0019    816          BBS     #7,LNMX$B_INDEX(R3),70$     ;BRANCH IF SPECIAL TRANSLATION
     51   55    D1  001E    817 20$:     CMPL    R5,R1                       ;BLOCK LINKED ON TABLE LIST?
          2C    12  0021    818          BNEQ    DELETE_LNMB                 ;BRANCH IF NOT TO LINKED TO
          05    23  0023    819          RSB                                 ;DELETE THE NAME BLOCK AND RETURN
                     0024    820
  50    05 A3    DE  0024    821 50$:     MOVAL   LNMX$T_XLATION+1(R3),R0     ;TRANSLATION STRING ADDRESS
          60    D5  0028    822          TSTL    (R0)                        ;IS ADDRESS PRESENT?
          04    13  002A    823          BEQL    60$                         ;NO ADDRESS SPECIFIED
          90    D4  002C    824          CLRL    @(R0)+                      ;CLEAR POINTER TO NAME
          70    D4  002E    825          CLRL    -(R0)                       ;CLEAR BACK POINTER
     53   04    C0  0030    826 60$:     ADDL    #LNMX$T_XLATION,R3          ;ADDRESS OF COUNTED TRANSLATION STRING
          DA    11  0033    827          BRB     10$                         ;PROCESS NEXT TRANSLATION
                     0035    828
81 8F    01 A3    91  0035    829 70$:     CMPB    LNMX$B_INDEX(R3),#LNMX$C_BACKPTR ;LOOK FOR BACK POINTER
          E8    13  003A    830          BEQL    50$                         ;PROCESS REFERENCE POINTER
82 8F    01 A3    91  003C    831          CMPB    LNMX$B_INDEX(R3),#LNMX$C_TABLE  ;LOOK FOR TABLE HEADER
```

```
                ED   12  0041   832          BNEQ    60$                          ;CONTINUE SCANNING TRANSLATIONS
                         0043   833          ASSUME  LNMB$L_FLINK,EQ,0
          81    55   D0  0043   834          MOVL    R5,(R1)+                     ;LINK ONTO R5 LIST
    61    05    A3   9E  0046   835          MOVAB   LNMX$T_XLATION+1(R3), -      ;STORE TABLE IDENTIFIER IN A HANDY PLACE
                         004A   836                  LNMB$L_BLINK-4(R1)
          55    71   DE  004A   837          MOVAL   -(R1),R5
                E1   11  004D   838          BRB     60$                          ;CONTINUE SCAN OF TRANSLATIONS
                         004F   839
                         004F   840 ;        .PAGE
```

```
                     004F    842          .SBTTL  DELETE_LNMB     - DELETE LOGICAL NAME BLOCK
                     004F    843 ;+
                     004F    844 ; DELETE_LNMB - DELETE LOGICAL NAME BLOCK
                     004F    845 ;
                     004F    846 ; THIS ROUTINE IS CALLED TO DELETE A LOGICAL NAME BLOCK.  STORAGE IS
                     004F    847 ; RETURNED TO THE APPROPRIATE ALLOCATION REGION AND QUOTA IS RETURNED
                     004F    848 ; TO THE APPROPRIATE TABLE HEADER(S).  IF THE LOGICAL NAME BLOCK IS FOR A
                     004F    849 ; SHAREABLE LOGICAL NAME TABLE, THEN THE OBJECT RIGHTS BLOCK ASSOCIATED WITH
                     004F    850 ; THE TABLE IS CLEANED UP BEFORE IT, TOGETHER WITH THE LNMB, ARE DELETED.
                     004F    851 ;
                     004F    852 ; INPUTS:
                     004F    853 ;
                     004F    854 ;       R1 = ADDRESS OF ENTRY TO BE DELETED.
                     004F    855 ;
                     004F    856 ;       IF THE ENTRY IS FOR A LOGICAL NAME TABLE, IT IS ASSUMED THAT THE ADDRESS
                     004F    857 ;       OF ITS TABLE HEADER IS STORED IN LNMB$L_BLINK.
                     004F    858 ;
                     004F    859 ;       IT IS ASSUMED THAT THE LOGICAL NAME MUTEX IS LOCKED FOR WRITE ACCESS,
                     004F    860 ;       AND THAT THE CALLER HAS THE PRIVILEGE OF DELETING THE LOGICAL NAME
                     004F    861 ;       BLOCK.
                     004F    862 ;
                     004F    863 ; OUTPUTS:
                     004F    864 ;
                     004F    865 ;       STORAGE IS RETURNED TO THE APPROPRIATE ALLOCATION REGION.
                     004F    866 ;       THE ORB ASSOCIATED WITH THE LNMB, IF THERE IS ONE, IS CLEANED UP.
                     004F    867 ;       QUOTA IS RETURNED TO THE APPROPRIATE TABLE HEADER(S).
                     004F    868 ;
                     004F    869 ;       R0, R1, R2, AND R3 ARE MODIFIED.
                     004F    870 ;-
                     004F    871
                     004F    872 DELETE_LNMB:                            ;DELETE LOGICAL NAME BLOCK
     50    51    D0  004F    873          MOVL    R1,R0                  ;ADDRESS OF LOGICAL NAME BLOCK
           03    E1  0052    874          BBC     #LNMB$V_TABLE,-        ;LOGICAL NAME TABLE?
  44 10 A0           0054    875                  LNMB$B_FLAGS(R0),40$   ;NO - GO RETURN QUOTA FOR THE ENTRY
                     0057    876
                     0057    877 ;
                     0057    878 ; THE LOGICAL NAME BLOCK BEING DEALLOCATED IS FOR A LOGICAL NAME TABLE.  TAKE
                     0057    879 ; THE FOLLOWING ACTIONS BEFORE DELETING THE LOGICAL NAME BLOCK:
                     0057    880 ;
                     0057    881 ; 1. REMOVE THE LOGICAL NAME TABLE ENTRY BEING DELETED FROM THE HIERARCHY OF
                     0057    882 ;    LOGICAL NAME TABLES BY MODIFYING THE CHILD POINTER OF THE LOGICAL NAME
                     0057    883 ;    TABLE'S PARENT TABLE, AND THE SIBLING POINTERS OF THE LOGICAL NAME TABLE'S
                     0057    884 ;    SIBLINGS AS REQUIRED.
                     0057    885 ;
                     0057    886 ; 2. RETURN TO THE PARENT LOGICAL NAME TABLE'S QUOTA HOLDER (WHICH MAY IN FACT
                     0057    887 ;    BE THE PARENT TABLE) ANY QUOTA EXPLICTELY ALLOCATED TO THE TABLE.
                     0057    888 ;
                     0057    889 ; 3. IF THE LOGICAL NAME TABLE IS SHAREABLE, CLEANUP THE OBJECT RIGHTS BLOCK
                     0057    890 ;    ASSOCIATED WITH THE TABLE. THE ORB ITSELF WAS ALLOCATED CONTIGIOUSLY WITH
                     0057    891 ;    THE LOGICAL NAME BLOCK AND WILL BE DELETED WHEN THE LOGICAL NAME BLOCK IS
                     0057    892 ;    ITSELF DELETED.
                     0057    893 ;
                     0057    894
 53    04 A0    D0  0057    895          MOVL    LNMB$L_BLINK(R0),R3    ;ADDRESS OF TABLE HEADER
 52    0D A3    D0  005B    896          MOVL    LNMTH$L_PARENT(R3),R2  ;ADDRESS OF PARENT'S TABLE HEADER
11 A2    53    D1  005F    897          CMPL    R3,LNMTH$L_CHILD(R2)   ;IS BLOCK THE IMMEDIATE CHILD OF PARENT?
           07    12  0063    898          BNEQ    10$                    ;NO  - GO FIND PRECEEDING SIBLING
```

```
                     15 A3   D0  0065   899          MOVL    LNMTH$L_SIBLING(R3),-   ;YES - BLOCK'S IMMEDIATE SIBLING BECOMES
                     11 A2       0068   900                  LNMTH$L_CHILD(R2)       ;       PARENT'S IMMEDIATE CHILD
                        15   11  006A   901          BRB     30$                     ;GO RETURN DEDUCTED QUOTA
                                 006C   902
                  51 11 A2   D0  006C   903 10$:     MOVL    LNMTH$L_CHILD(R2),R1    ;ADDRESS OF PARENT'S IMMEDIATE CHILD
                15 A1    53   D1  0070   904 15$:     CMPL    R3,LNMTH$L_SIBLING(R1)  ;DOES THE SIBLING PRECEED THE BLOCK?
                         06   13  0074   905          BEQL    20$                     ;YES - GO UNHOOK IT FROM LIST
                  51 15 A1   D0  0076   906          MOVL    LNMTH$L_SIBLING(R1),R1  ;NO  - RETRIEVE ADDRESS OF NEXT SIBLING
                         F4   11  007A   907          BRB     15$                     ;       AND CONTINUE SEARCH
                                 007C   908
                     15 A3   D0  007C   909 20$:     MOVL    LNMTH$L_SIBLING(R3),-   ;UNHOOK LOGIAL NAME BLOCK FROM THE LIST
                     15 A1       007F   910                  LNMTH$L_SIBLING(R1)     ;OF SIBLINGS
                                 0081   911
                  52 19 A2   D0  0081   912 30$:     MOVL    LNMTH$L_QTABLE(R2),R2   ;RETRIEVE PARENT'S QUOTA HOLDER
                     1D A3   C0  0085   913          ADDL2   LNMTH$L_BYTESLM(R3),-   ;RETURN ANY DEDUCTED QUOTA TO PARENT'S
                     21 A2       0088   914                  LNMTH$L_BYTES(R2)       ;QUOTA HOLDER
                                 008A   915
                  51 05 A3   D0  008A   916          MOVL    LNMTH$L_ORB(R3),R1      ;RETRIEVE TABLE'S OBJECT RIGHTS BLOCK
                         0B   13  008E   917          BEQL    40$                     ;SKIP ORB CLEANUP IF THERE ISN'T ONE
                         50   DD  0090   918          PUSHL   R0                      ;SAVE LNMB ADDRESS
            00000000'EF      16  0092   919          JSB     EXE$CLEANUP_ORB         ;CLEANUP THE ORB
                     50 8ED0     0098   920          POPL    R0                      ;RESTORE LNMB ADDRESS
                                 009B   921
                                 009B   922 ;
                                 009B   923 ; FINALLY, RETURN THE SIZE OF THE LOGICAL NAME (OR LOGICAL NAME TABLE) TO THE
                                 009B   924 ; QUOTA HOLDER OF THE CONTAINING TABLE, AND THEN PERFORM THE DELETION OF THE
                                 009B   925 ; ACTUAL BLOCK.
                                 009B   926 ;
                                 009B   927
                  51 08 A0   3C  009B   928 40$:     MOVZWL  LNMB$W_SIZE(R0),R1      ;SIZE OF LOGICAL NAME BLOCK TO RETURN
                  52 0C A0   D0  009F   929          MOVL    LNMB$L_TABLE(R0),R2     ;ADDRESS OF CONTAINING TABLE'S HEADER
                                 00A3   930
                                 00A3   931 ;
                                 00A3   932 ; BUMP THE APPROPRIATE DIRECTORY SEQUENCE NUMBER IF THE CONTAINING TABLE
                                 00A3   933 ; IS ONE OF THE DIRECTORY TABLES.
                                 00A3   934 ;
                                 00A3   935
                        01   E1  00A3   936          BBC     #LNMTH$V_DIRECTORY,-    ;DIRECTORY TABLE?
                     12 62       00A5   937                  LNMTH$B_FLAGS(R2),60$   ;OKAY IF NOT
                  08 52 1F   E0  00A7   938          BBS     #31,R2,50$              ;BRANCH IF SYSTEM DIRECTORY
            00000000'9F      D6  00AB   939          INCL    @#CTL$GL_LNMDIRSEQ      ;BUMP PROCESS DIRECTORY COUNTER
                         06   11  00B1   940          BRB     60$
            00000000'9F      D6  00B3   941 50$:     INCL    @#LNM$GL_SYSDIRSEQ      ;BUMP SYSTEM DIRECTORY COUNTER
                                 00B9   942
                  52 19 A2   D0  00B9   943 60$:     MOVL    LNMTH$L_QTABLE(R2),R2   ;ADDRESS OF QUOTA HOLDER OF TABLE
                  21 A2 51   C0  00BD   944          ADDL2   R1,LNMTH$L_BYTES(R2)    ;RETURN SIZE OF LOGICAL NAME
                        00CE   31  00C1   945          BRW     LNM$DELBLK              ;DELETE LOGICAL NAME BLOCK AND RETURN
                                 00C4   946
                                 00C4   947 ;        .PAGE
```

```
                     00C4   949          .SBTTL  DELETE_NAMES    - SCAN HASH TABLE AND DELETE NAMES
                     00C4   950 ;+
                     00C4   951 ; DELETE_NAMES - SCAN HASH TABLE AND DELETE NAMES
                     00C4   952 ;
                     00C4   953 ; THIS ROUTINE SCANS A HASH TABLE AND DELETES ALL OF THE NAMES IN A SPECIFIED
                     00C4   954 ; TABLE WITH AN ACCESS MODE GREATER THAN OR EQUAL TO THE SPECIFIED ACCESS MODE.
                     00C4   955 ;
                     00C4   956 ; THIS ROUTINE IS SET UP SO THAT IT WILL NEVER DELETE A DIRECTORY.
                     00C4   957 ;
                     00C4   958 ; INPUTS:
                     00C4   959 ;
                     00C4   960 ;       R1 = ADDRESS OF TABLE HEADER OR 0 FOR ALL TABLES.
                     00C4   961 ;       R2 = ACCESS MODE.
                     00C4   962 ;       R3 = ADDRESS OF HASH TABLE TO BE SCANNED.
                     00C4   963 ;
                     00C4   964 ;       IT IS ASSUMED THAT THE LOGICAL NAME MUTEX IS LOCKED FOR WRITE ACCESS,
                     00C4   965 ;       AND THAT THE CALLER HAS THE PRIVILEGE OF DELETING THE LOGICAL NAME
                     00C4   966 ;       TABLE ENTRIES WITHIN THE SPECIFIED TABLE.
                     00C4   967 ;
                     00C4   968 ; OUTPUTS:
                     00C4   969 ;
                     00C4   970 ;       ENTRIES ARE REMOVED FROM THEIR RESPECTIVE LOGICAL NAME TABLES AND THE
                     00C4   971 ;       STORAGE IS RETURNED TO THE APPROPRIATE ALLOCATION REGION.
                     00C4   972 ;
                     00C4   973 ;       R1, R2, R3, R4, and R5 ARE MODIFIED.
                     00C4   974 ;-
                     00C4   975
                     00C4   976 DELETE_NAMES:
          55    D4   00C4   977          CLRL    R5                              ;ZERO LINKED LIST OF TABLE HEADERS
       54 51    D0   00C6   978          MOVL    R1,R4                           ;ADDRESS OF TABLE HEADER
       50 63    CE   00C9   979          MNEGL   LNMHSH$L_MASK(R3),R0            ;MAXIMUM BUCKET NUMBER
    53  0C A3    DE   00CC   980          MOVAL   LNMHSH$K_BUCKET(R3),R3          ;BUCKET ADDRESS
       51 83    D0   00D0   981 10$:     MOVL    (R3)+,R1                        ;BEGINNING OF HASH CHAIN
          28    13   00D3   982          BEQL    40$                             ;EMPTY CHAIN
          54    D5   00D5   983 20$:     TSTL    R4                              ;SPECIFIC TABLE
          06    13   00D7   984          BEQL    25$                             ;NOPE
    0C A1 54    D1   00D9   985          CMPL    R4,LNMB$L_TABLE(R1)             ;RIGHT TABLE?
          19    12   00DD   986          BNEQ    30$                             ;BRANCH IF TABLES DON'T MATCH
    0B A1 52    91   00DF   987 25$:     CMPB    R2,LNMB$B_ACMODE(R1)            ;ACCESS MODE COMPATIBLE?
          13    1A   00E3   988          BGTRU   30$                             ;BRANCH IF NOT
          04    E0   00E5   989          BBS     #LNMB$V_NODELETE,-              ;DIRECTORY (NODELETE WILL BE SET)?
       0E 10 A1           00E7   990                  LNMB$B_FLAGS(R1),30$        ;BRANCH IF DIRECTORY, NEVER DELETE ONE
          1F    BB   00EA   991          PUSHR   #^M<R0,R1,R2,R3,R4>             ;SAVE SOME REGISTERS
04 AE    04 A1    D0   00EC   992          MOVL    LNMB$L_BLINK(R1),4(SP)          ;SAVE ADDRESS OF PREVIOUS BLOCK
          FF0C    30   00F1   993          BSBW    DELETE_ENTRY                    ;DELETE LOGICAL NAME BLOCK
          0B      10   00F4   994          BSBB    DELETE_TABLE                    ;DELETE TABLE IF CURRENT LNMB IS ONE
          1F      BA   00F6   995          POPR    #^M<R0,R1,R2,R3,R4>             ;RESTORE REGISTERS
       51 61    D0   00F8   996 30$:     MOVL    LNMB$L_FLINK(R1),R1             ;MOVE TO NEXT LNMB IN HASH BUCKET
          D8    12   00FB   997          BNEQ    20$                             ;BRANCH IF IT EXISTS
       D0 50    F5   00FD   998 40$:     SOBGTR  R0,10$                          ;BUMP TO NEXT HASH BUCKET
                05   0100   999          RSB
                     0101  1000
                     0101  1001 ;        .PAGE
```

```
                     0101  1003          .SBTTL  DELETE_TABLE    - DELETE A LOGICAL NAME TABLE
                     0101  1004 ;+
                     0101  1005 ; DELETE_TABLE - DELETE A LOGICAL NAME TABLE
                     0101  1006 ;
                     0101  1007 ; THIS ROUTINE TAKES A TABLE HEADER AND DELETES IT. TO DELETE A TABLE HEADER,
                     0101  1008 ; FIRST ALL OF THE CHILDREN OF THE TABLE ARE DELETED BY RECURSIVELY CALLING
                     0101  1009 ; THIS ROUTINE, THEN ALL OF THEN NAMES DEFINED WITHIN THE TABLE ARE DELETED,
                     0101  1010 ; AND FINALLY THE LOGICAL NAME BLOCK CONTAINING THE TABLE HEADER ITSELF IS
                     0101  1011 ; DELETED.
                     0101  1012 ;
                     0101  1013 ; INPUTS:
                     0101  1014 ;
                     0101  1015 ;       R5 = ADDRESS OF LIST OF LOGICAL NAME BLOCK CONTAINING TABLE HEADERS TO
                     0101  1016 ;            BE DELETED.
                     0101  1017 ;
                     0101  1018 ;       IT IS ASSUMED THAT THE ADDRESS OF THE TABLE HEADER IS STORED WITHIN
                     0101  1019 ;       LNMB$L_BLINK(R5).
                     0101  1020 ;
                     0101  1021 ;       IT IS ASSUMED THAT THE LOGICAL NAME MUTEX IS LOCKED FOR WRITE ACCESS,
                     0101  1022 ;       AND THAT THE CALLER HAS THE PRIVILEGE OF DELETING THE LOGICAL NAME
                     0101  1023 ;       TABLE.
                     0101  1024 ;
                     0101  1025 ; OUTPUTS:
                     0101  1026 ;
                     0101  1027 ;       ALL OF THE NAMES DEFINED WITHIN THE TABLE HEADER ARE DELETED.
                     0101  1028 ;       ALL OF THE CHILDREN OF THE TABLE HEADER ARE DELETED (THIS INVOLVES
                     0101  1029 ;       DELETING ALL OF THEIR CHILDREN, AND ALL OF THE NAMES DEFINED WITHIN
                     0101  1030 ;       THEM ETC...).
                     0101  1031 ;       STORAGE AND QUOTA IS RETURNED.
                     0101  1032 ;
                     0101  1033 ;       R1, R2, R3, R4, and R5 ARE MODIFIED.
                     0101  1034 ;-
                     0101  1035
                     0101  1036 DELETE_TABLE:
    50    55    D0   0101  1037          MOVL    R5,R0                   ;ADDRESS OF CURRENT TABLE HEADER'S LNMB
          04    12   0104  1038          BNEQ    10$                     ;BRANCH IF ONE PRESENT
    50    01    D0   0106  1039          MOVL    #SS$_NORMAL,R0          ;RETURN SUCCESS WHEN ALL DONE
                05   0109  1040          RSB
                     010A  1041
                     010A  1042 ;
                     010A  1043 ; DELETE ALL OF THE CHILDREN OF THE CURRENT TABLE HEADER.
                     010A  1044 ;
                     010A  1045
51    04    A0   D0   010A  1046 10$:    MOVL    LNMB$L_BLINK(R0),R1     ;ADDRESS OF TABLE HEADER
52    11    A1   D0   010E  1047          MOVL    LNMTH$L_CHILD(R1),R2    ;ADDRESS OF CHILD TABLE
          09   13   0112  1048          BEQL    20$                     ;BRANCH IF NO CHILD
51    09    A2   D0   0114  1049          MOVL    LNMTH$L_NAME(R2),R1     ;ADDRESS OF NAME BLOCK OF CHILD
        FEE5   30   0118  1050          BSBW    DELETE_ENTRY            ;DELETE THE CHILD (PUT IT ON THE LIST)
          E4   11   011B  1051          BRB     DELETE_TABLE            ;CONTINUE THE DELETION PROCESS
                     011D  1052
                     011D  1053 ;
                     011D  1054 ; DELETE ALL OF THE NAMES DEFINED WITHIN THE CURRENT TABLE HEADER.
                     011D  1055 ;
                     011D  1056
          60    DD   011D  1057 20$:    PUSHL   LNMB$L_FLINK(R0)        ;UPDATE AND SAVE LIST POINTER
          50    DD   011F  1058          PUSHL   R0                      ;SAVE CURRENT LOGICAL NAME BLOCK ADDRESS
53    01    A1   D0   0121  1059          MOVL    LNMTH$L_HASH(R1),R3     ;HASH TABLE ADDRESS
```

```
                0125  1060          CLRL    R2                      ;ALL ACCESS MODES
FF9A 30         0127  1061          BSBW    DELETE_NAMES            ;DELETE NAMES IN THIS TABLE
  22 BA         012A  1062          POPR    #^M<R1,R5>              ;RESTORE NAME BLOCK ADDRESS AND LIST PTR
FF20 30         012C  1063          BSBW    DELETE_LNMB             ;DELETE THE CURRENT LOGICAL NAME BLOCK
  D0 11         012F  1064          BRB     DELETE_TABLE            ;ITERATE UNTIL LIST IS EMPTY
                0131  1065
                0131  1066 ;        .PAGE
```

```
                     0131  1068          .SBTTL  LNM$CHECK_PROT  - CHECK ACCESS TO A LOGICAL NAME TABLE
                     0131  1069 ;+
                     0131  1070 ; LNM$CHECK_PROT - CHECK ACCESS TO A LOGICAL NAME TABLE
                     0131  1071 ;
                     0131  1072 ; THIS ROUTINE IS CALLED TO CHECK ACCESS TO A LOGICAL NAME TABLE, USING ITS
                     0131  1073 ; OBJECT RIGHTS BLOCK.
                     0131  1074 ;
                     0131  1075 ; ASSUMPTION: THE LOGICAL NAME TABLE HAS AN ASSOCIATED OBJECT RIGHTS BLOCK
                     0131  1076 ;               IE - LNMTH$L_CHP(R1) .= 0!
                     0131  1077 ;
                     0131  1078 ; CURRENTLY, ONLY SOGW PROTECTION IS ACTUALLY CHECKED.   ACCESS MODE
                     0131  1079 ; CHECKING IS HANDLED BY THE CALLING ROUTINE AND ALL OTHER SECURITY
                     0131  1080 ; CHECKS ARE NYI.
                     0131  1081 ;
                     0131  1082 ; INORDER TO PROVIDE COMPATIBLE ACCESS TO THE GROUP AND SYSTEM LOGICAL NAME
                     0131  1083 ; TABLES ACROSS RELEASES, THE FOLLOWING ALGORITHM (WHICH WILL ALLOW ACCESS
                     0131  1084 ; UNDER SPECIAL SETS OF CIRCUMSTANCES TO THESE TABLES EVEN IF ACCESS IS DENIED
                     0131  1085 ; BY SOGW PROTECTION) IS IMPLEMENTED:
                     0131  1086 ;
                     0131  1087 ; IF SOGW PROTECTION
                     0131  1088 ; THEN
                     0131  1089 ;     RETURN SUCCESS
                     0131  1090 ; ELSE
                     0131  1091 ;     IF OTHER THAN R OR W ACCESS IS REQUESTED
                     0131  1092 ;     THEN
                     0131  1093 ;         RETURN FAILURE
                     0131  1094 ;     ELSE
                     0131  1095 ;         IF GROUP LOGICAL NAME TABLE
                     0131  1096 ;         THEN
                     0131  1097 ;             IF GRPNAM AND TABLE IS GROUP TABLE FOR THE CALLER
                     0131  1098 ;             THEN
                     0131  1099 ;                 RETURN SUCCESS
                     0131  1100 ;             ELSE
                     0131  1101 ;                 RETURN FAILURE
                     0131  1102 ;         ELSE
                     0131  1103 ;             IF SYSTEM LOGICAL NAME TABLE AND SYSNAM
                     0131  1104 ;             THEN
                     0131  1105 ;                 RETURN SUCCESS
                     0131  1106 ;             ELSE
                     0131  1107 ;                 RETURN FAILURE
                     0131  1108 ;
                     0131  1109 ; INPUTS:
                     0131  1110 ;
                     0131  1111 ;       R1 = ADDRESS OF LOGICAL NAME TABLE HEADER.
                     0131  1112 ;       R2 = ACCESS MASK TO USE FOR THE CHECK.
                     0131  1113 ;       R4 = CURRENT PCB ADDR
                     0131  1114 ;
                     0131  1115 ; OUTPUTS:
                     0131  1116 ;
                     0131  1117 ;       R0 = RETURN STATUS FROM THE EXE$CHKPRO SUBROUTINE.
                     0131  1118 ;            (SUCCESS IN TWO SPECIAL CASES - SEE ABOVE)
                     0131  1119 ;       R2 IS DESTROYED. ALL OTHER REGISTERS PRESERVED.
                     0131  1120 ;-
                     0131  1121
                     0131  1122 LNM$CHECK_PROT::                         ;CHECK ACCESS TO A LOGICAL NAME TABLE
          53   DD    0131  1123          PUSHL   R3                      ;SAVE REGISTERS
7E        51   7D    0133  1124          MOVQ    R1,-(SP)
```

```
                                                I 15
50   008C C4    D0  0136  1125          MOVL    PCB$L_ARB(R4),R0        ;RETRIEVE AGENT RIGHTS BLOCK
 51    05 A1    D0  013B  1126          MOVL    LNMTH$L_ORB(R1),R1      ;RETRIEVE OBJECT RIGHTS BLOCK
                    013F  1127
         7E    D4  013F  1128          CLRL    -(SP)                   ;CLEAR ACCESS MODE FIELD OF CHPCTL
         05    DD  0141  1129          PUSL    #<CHPCTL$M_READ!-       ;SET READ AND USEREADALL BITS IN
                    0143  1130                  CHPCTL$M_USEREADALL>    ;CHPCTL FLAGS FIELD
 52    FE 8F    93  0143  1131          BITB    #^C<ARM$M_READ>,R2      ;OTHER THAN READ ACCESS CHECK REQUESTED?
         03    13  0147  1132          BEQL    5$                      ;BRANCH IF NOT
    6E   06    8C  0149  1133          XORB2   #<CHPCTL$M_WRITE!-      ;OTHERWISE SET WRITE AND CLEAR
                    014C  1134                  CHPCTL$M_USEREADALL>,(SP);USEREADLL BITS IN CHPCTL FLAGS FIELD
         52    DD  014C  1135  5$:     PUSHL   R2                      ;SET ACCESS REQUESTED IN CHPCTL
                    014E  1136
    52   5E    D0  014E  1137          MOVL    SP,R2                   ;MOVE CHPCTL ADDRESS INTO PROPER REG
         53    D4  0151  1138          CLRL    R3                      ;NO CHPRET ACCESS BLOCK REQUIRED
00000000'GF    16  0153  1139          JSB     G^EXE$CHKPRO_INT        ;CHECK THE PROTECTION
 5E   0C A2    9E  0159  1140          MOVAB   CHPCTL$C_LENGTH(R2),SP  ;REMOVE CHPCTL BLOCK FROM THE STACK
    51   8E    7D  015D  1141          MOVQ    (SP)+,R1                ;RESTORE REGISTERS
         53 8ED0  0160  1142          POPL    R3
      2B 50    E8  0163  1143          BLBS    R0,30$                  ;RETURN IF OK TO ACCESS
                    0166  1144
 52   FC 8F    93  0166  1145          BITB    #^C<ARM$M_READ!-        ;OTHER THAN R OR W ACCESS REQUESTED?
                    016A  1146                  ARM$M_WRITE>,R2
         25    12  016A  1147          BNEQ    30$                     ;IF SO THEN RETURN FAILURE
                    016C  1148
         02    E1  016C  1149          BBC     #LNMTH$V_GROUP,-        ;IS THIS A GROUP LOGICAL NAME TABLE?
      14 61        016E  1150                  LNMTH$B_FLAGS(R1),10$   ;BRANCH IF IT ISN'T ONE
                    0170  1151          IFNPRIV GRPNAM,30$              ;RETURN FAILURE IF CALLER LACKS GRPNAM
 52   05 A1    D0  0176  1152          MOVL    LNMTH$L_ORB(R1),R2      ;RETRIEVE ADDRESS OF OBJECT RIGHTS BLOCK
   00BE C4    B1  017A  1153          CMPW    PCB$L_UIC+2(R4),-       ;IS THIS THE CALLER'S GROUP LOGICAL
      02 A2        017E  1154                  ORB$L_OWNER+2(R2)       ;NAME TABLE?
         0C    13  0180  1155          BEQL    20$                     ;RETURN SUCCESS IF IT
         0D    11  0182  1156          BRB     30$                     ;RETURN FAILURE IF IT ISN'T
                    0184  1157
         03    E1  0184  1158  10$:    BBC     #LNMTH$V_SYSTEM,-       ;IS THIS THE SYSTEM LOGICAL NAME TABLE
      09 61        0186  1159                  LNMTH$B_FLAGS(R1),30$   ;RETURN FAILURE IF IT ISN'T
                    0188  1160          IFNPRIV SYSNAM,30$              ;RETURN FAILURE IF CALLER LACKS SYSNAM
                    018E  1161
50   01    D0  018E  1162  20$:    MOVL    #SS$_NORMAL,R0          ;CHANGE STATUS OF ROUTINE TO SUCCESS
         05    0191  1163  30$:    RSB                             ;RETURN STATUS
                    0192  1164
                    0192  1165  ;       .PAGE
```

```
                                                J 15
                     0192  1167          .SBTTL  LNM$DELBLK      - DELETE P1 OR S0 PAGED BLOCK
                     0192  1168 ;+
                     0192  1169 ; LNM$DELBLK - DELETE P1 OR S0 PAGED BLOCK
                     0192  1170 ;
                     0192  1171 ; THIS ROUTINE IS CALLED TO DELETE A CHUNK OF P1 OR S0 PAGED MEMORY.
                     0192  1172 ;
                     0192  1173 ; INPUTS:
                     0192  1174 ;
                     0192  1175 ;       R0 = ADDRESS OF ENTRY TO BE DELETED.
                     0192  1176 ;
                     0192  1177 ; OUTPUTS:
                     0192  1178 ;
                     0192  1179 ;       NONE
                     0192  1180 ;
                     0192  1181 ;       STORAGE IS RETURNED TO THE APPROPRIATE ALLOCATION REGION.
                     0192  1182 ;
                     0192  1183 ;       R0, R1, R2, AND R3 ARE MODIFIED.
                     0192  1184 ;-
                     0192  1185
                     0192  1186 LNM$DELBLK::                                ;DELETE PAGED BLOCK
 51    08 A0    3C    0192  1187          MOVZWL  LNMB$W_SIZE(R0),R1        ;SIZE OF BLOCK
 06 50    1F    E0    0196  1188          BBS     #31,R0,10$                ;IF SET, SYSTEM SPACE TABLE
00000000'9F    17    019A  1189          JMP     @#EXE$DEAP1               ;DEALLOCATE PROCESS DYNAMIC MEMORY
00000000'9F    17    01A0  1190 10$:     JMP     @#EXE$DEAPAGED            ;DEALLOCATE SYSTEM PAGED MEMORY
                     01A6  1191
                     01A6  1192 ;        .PAGE
```

```
                        01A6  1194          .SBTTL  LNM$DELETE      - DELETE LOGICAL NAME TABLE ENTRY
                        01A6  1195 ;+
                        01A6  1196 ; LNM$DELETE - DELETE LOGICAL NAME TABLE ENTRY
                        01A6  1197 ;
                        01A6  1198 ; THIS ROUTINE IS CALLED TO DELETE A LOGICAL NAME TABLE ENTRY, RETURN ITS
                        01A6  1199 ; STORAGE TO THE APPROPRIATE ALLOCATION REGION, AND RETURN ITS QUOTA.
                        01A6  1200 ; IF THE NAME IS A TABLE HEADER, THEN ALL NAMES IN THE TABLE ARE ALSO
                        01A6  1201 ; DELETED AND ANY TABLES OF WHICH THIS TABLE IS THE PARENT ARE DELETED.
                        01A6  1202 ;
                        01A6  1203 ; A PRIVILEGE ERROR WILL ALWAYS BE RETURNED IF AN ATTEMPT IS MADE TO DELETE
                        01A6  1204 ; A DIRECTORY.
                        01A6  1205 ;
                        01A6  1206 ; INPUTS:
                        01A6  1207 ;
                        01A6  1208 ;       R1 = ADDRESS OF ENTRY TO BE DELETED.
                        01A6  1209 ;
                        01A6  1210 ;       IT IS ASSUMED THAT THE LOGICAL NAME MUTEX IS LOCKED FOR WRITE ACCESS,
                        01A6  1211 ;       AND THAT THE CALLER HAS THE PRIVILEGE OF DELETING THE LOGICAL NAME
                        01A6  1212 ;       BLOCK.
                        01A6  1213 ;
                        01A6  1214 ; OUTPUTS:
                        01A6  1215 ;
                        01A6  1216 ;       R0 = SS$_NOPRIV IF THE LOGICAL NAME TABLE ENTRY IS A DIRECTORY.
                        01A6  1217 ;
                        01A6  1218 ;       R0 = SS$_SUCCESS
                        01A6  1219 ;
                        01A6  1220 ;       THE ENTRY IS REMOVED FROM ITS RESPECTIVE LOGICAL NAME TABLE AND THE
                        01A6  1221 ;       STORAGE IS RETURNED TO THE APPROPRIATE ALLOCATION REGION. IF THE ENTRY
                        01A6  1222 ;       HAS AN ASSOCIATED MAILBOX UCB ADDRESS, THEN THE LINKAGE FROM THE UCB
                        01A6  1223 ;       TO THE LOGICAL NAME ENTRY IS CLEARED.
                        01A6  1224 ;
                        01A6  1225 ;       R1, R2, AND R3 ARE MODIFIED.
                        01A6  1226 ;-
                        01A6  1227
                        01A6  1228
                        01A6  1229 LNM$DELETE:                                  ;DELETE LOGICAL NAME TABLE ENTRY
     04     E1          01A6  1230          BBC     #LNMB$V_NODELETE,-          ;DIRECTORY (NODELETE WILL BE SET)?
05 10 A1                01A8  1231                  LNMB$B_FLAGS(R1),10$        ;IF NOT THEN GO DELETE THE ENTRY
 50     24     3C       01AB  1232          MOVZWL  #SS$_NOPRIV,R0              ;OTHERWISE RETURN A PRIVILEGE VIOLATION
        0E     11       01AE  1233          BRB     20$
                        01B0  1234
 7E     54     7D       01B0  1235 10$:     MOVQ    R4,-(SP)                    ;SAVE REGISTERS R4 AND R5
        55     D4       01B3  1236          CLRL    R5                          ;CLEAR LINKED LIST OF TABLE HEADERS
    FE48        30       01B5  1237          BSBW    DELETE_ENTRY                ;DELETE THE LOGICAL NAME TABLE ENTRY
    FF46        30       01B8  1238          BSBW    DELETE_TABLE                ;DELETE THE TABLE HEADER IF IT IS ONE
 54     8E     7D       01BB  1239          MOVQ    (SP)+,R4                    ;RESTORE REGISTERS R4 AND R5
        05     05       01BE  1240 20$:     RSB                                 ;RETURN STATUS
                        01BF  1241
                        01BF  1242 ;        .PAGE
```

```
                     01BF  1244          .SBTTL  LNM$DELETE_LNMB - DELETE LOGICAL NAME TABLE ENTRY PLUS ALIASES
                     01BF  1245 ;+
                     01BF  1246 ; LNM$DELETE_LNMB - DELETE LOGICAL NAME TABLE ENTRY PLUS ALIASES
                     01BF  1247 ;
                     01BF  1248 ; THIS ROUTINE IS CALLED TO DELETE A LOGICAL NAME TABLE ENTRY TOGETHER WITH ALL
                     01BF  1249 ; OF ITS OUTER ACCESS MODE ALIASES. STORAGE FOR THE DELETED ENTRIES IS RETURNED
                     01BF  1250 ; TO THE APPROPRIATE ALLOCATION REGION, AND QUOTA IS RETURNED AS WELL. IF ANY
                     01BF  1251 ; OF THE ENTRIES DELETED ARE LOGICAL NAME TABLES, THEN ALL NAMES WITHIN THE
                     01BF  1252 ; TABLE ARE ALSO DELETED AS WELL AS ANY TABLES OF WHICH THIS TABLE IS THE
                     01BF  1253 ; PARENT OF.
                     01BF  1254 ;
                     01BF  1255 ; INPUTS:
                     01BF  1256 ;
                     01BF  1257 ;       R1 = ADDRESS OF LOGICAL NAME TABLE ENTRY
                     01BF  1258 ;
                     01BF  1259 ;       IT IS ASSUMED THAT THE LOGICAL NAME MUTEX IS LOCKED FOR WRITE ACCESS,
                     01BF  1260 ;       AND THAT THE CALLER HAS THE PRIVILEGE OF DELETING THE LOGICAL NAME
                     01BF  1261 ;       TABLE ENTRY.
                     01BF  1262 ;
                     01BF  1263 ; OUTPUTS:
                     01BF  1264 ;
                     01BF  1265 ;       R0 = SS$_NOPRIV IF THE LOGICAL NAME TABLE ENTRY IS A DIRECTORY.
                     01BF  1266 ;
                     01BF  1267 ;            SS$_SUCCESS.
                     01BF  1268 ;
                     01BF  1269 ;            SS$_NOLOGNAM IF THERE ARE NO SUCH LOGICAL NAMES.
                     01BF  1270 ;
                     01BF  1271 ;       THE ENTRIES ARE REMOVED FROM THEIR LOGICAL NAME TABLE AND THE STORAGE
                     01BF  1272 ;       IS RETURNED TO THE APPROPRIATE ALLOCATION REGION. IF ANY OF THE ENTRIES
                     01BF  1273 ;       HAS AN ASSOCIATED MAILBOX UCB OR VOLUME UCB ADDRESS, THEN THE LINKAGE
                     01BF  1274 ;       FROM THE UCB TO THE LOGICAL NAME TABLE ENTRY IS CLEARED BY THIS
                     01BF  1275 ;       ROUTINE'S CALLER.
                     01BF  1276 ;
                     01BF  1277 ;       R1, R2, AND R3 ARE MODIFIED.
                     01BF  1278 ;-
                     01BF  1279
                     01BF  1280 LNM$DELETE_LNMB::                            ;DELETE TABLE ENTRY AND ALL ALIASES
          7E    D4   01BF  1281          CLRL    -(SP)                      ;INITIALIZE THREAD TO ZERO
     0C A1    DD   01C1  1282          PUSHL   LNMB$L_TABLE(R1)           ;ADDRESS OF TABLE HEADER
     12 A1    9F   01C4  1283          PUSHAB  LNMB$T_NAME+1(R1)          ;DESCRIPTOR OF LOGICAL NAME
7E   11 A1    9A   01C7  1284          MOVZBL  LNMB$T_NAME(R1),-(SP)
00000403 8F    DD   01CB  1285          PUSHL   #<NT_M_MODIFY @ 8+-        ;HASH BUCKET MIGHT BE MODIFIED
                     01D1  1286                  PSL$C_USER>                ;SEARCH ACCESS MODE
          51    DD   01D1  1287          PUSHL   R1                         ;SAVE ADDRESS OF INNERMOST LNMB
                     01D3  1288
          03    91   01D3  1289          CMPB    #PSL$C_USER,-              ;IS INNERMOST LOGICAL NAME TABLE ENTRY
     0B A1         01D5  1290                  LNMB$B_ACMODE(R1)          ;AN USER ACCESS MODE NAME?
          07    13   01D7  1291          BEQL    10$                        ;IF SO, THEN IT IS ONLY ENTRY TO DELETE
53   04 AE    DE   01D9  1292          MOVAL   4(SP),R3                   ;ELSE SETUP TO FIND OUTERMOST MODE ENTRY
       02EB    30   01DD  1293          BSBW    LNM$PRESEARCH              ;AND GO FIND IT
                     01E0  1294
          51    DD   01E0  1295 10$:     PUSHL   R1                         ;SAVE ADDRESS OF CURRENT TABLE ENTRY
          C2    10   01E2  1296          BSBB    LNM$DELETE                 ;DELETE CURRENT TABLE ENTRY
          51  8ED0   01E4  1297          POPL    R1                         ;RESTORE ADDRESS OF DELETED ENTRY
     11 50    E9   01E7  1298          BLBC    R0,30$                     ;RETURN ANY ERROR ON DELETION
                     01EA  1299
   6E    51    D1   01EA  1300          CMPL    R1,(SP)                    ;WAS INNERMOST ENTRY JUST DELETED?
```

```
                09   13  01ED  1301          BEQL    20$                     ;IF SO THEN GO RETURN SUCCESS
          53  04 AE   DE  01EF  1302          MOVAL   4(SP),R3                ;ELSE PREPARE TO SEARCH FOR NEXT ENTRY
            02FF  30  01F3  1303          BSBW    LNM$CONTSEARCH          ;SEARCH FOR NEXT ENTRY TO DELETE
              E8  11  01F6  1304          BRB     10$                     ;ELSE GO SEE IF ENTRY CAN BE DELETED
                      01F8  1305
           50   01   D0  01F8  1306 20$:    MOVL    #SS$_NORMAL,R0          ;SUCCESS
           5E   18   C0  01FB  1307 30$:    ADDL2   #NT_R_LENGTH+4,SP       ;REMOVE STORAGE FROM STACK
                     05  01FE  1308          RSB                             ;RETURN STATUS
                         01FF  1309
                         01FF  1310 ;       .PAGE
```

```
                     01FF  1312          .SBTTL  LNM$DELETE_HASH - DELETE ALL ENTRIES IN A HASH TABLE
                     01FF  1313 ;+
                     01FF  1314 ; LNM$DELETE_HASH - DELETE ALL ENTRIES IN A HASH TABLE
                     01FF  1315 ;
                     01FF  1316 ; THIS ROUTINE IS CALLED TO DELETE ALL NAMES WITH A SPECIFIED OR GREATER
                     01FF  1317 ; ACCESS MODE FROM A SPECIFIED HASH TABLE.
                     01FF  1318 ;
                     01FF  1319 ; INPUTS:
                     01FF  1320 ;
                     01FF  1321 ;       R2 = ACCESS MODE.
                     01FF  1322 ;       R3 = ADDRESS OF HASH TABLE.
                     01FF  1323 ;       R4 = PCB ADDRESS.
                     01FF  1324 ;
                     01FF  1325 ;       THE LOGICAL NAME MUTEX WILL BE LOCKED FOR WRITE ACCESS,
                     01FF  1326 ;       AND THE CALLER IS ASSUMED TO HAVE THE PRIVILEGE OF DELETING
                     01FF  1327 ;       THE LOGICAL NAME TABLE ENTRIES WITHIN THE SPECIFIED TABLE.
                     01FF  1328 ;
                     01FF  1329 ; OUTPUTS:
                     01FF  1330 ;
                     01FF  1331 ;       THE APPROPRIATE HASH TABLE IS SCANNED AND ALL NAMES
                     01FF  1332 ;       WITH AN ACCESS MODE GREATER THAN OR EQUAL TO
                     01FF  1333 ;       THE SPECIFIED ACCESS MODE ARE DELETED.  QUOTA IS RETURNED.
                     01FF  1334 ;
                     01FF  1335 ;       R1, R2, AND R3 ARE MODIFIED.
                     01FF  1336 ;-
                     01FF  1337
                     01FF  1338 LNM$DELETE_HASH::
                     01FF  1339          SAVIPL                          ;SAVE CURRENT IPL ON STACK
7E     54    7D      0202  1340          MOVQ    R4,-(SP)                ;SAVE REGISTERS
    056A     30      0205  1341          BSBW    LNM$LOCKW               ;LOCK LOGICAL NAME MUTEX FOR W'ITING
      51     D4      0208  1342          CLRL    R1                      ;NO SPECIFIC TABLE HEADER
    FEB7     30      020A  1343          BSBW    DELETE_NAMES            ;DELETE THE NAMES WITHIN THE TABLE
54     8E    7D      020D  1344          MOVQ    (SP)+,R4                ;RESTORE REGISTERS
    0567     30      0210  1345          BSBW    LNM$UNLOCK              ;UNLOCK THE LOGICAL NAME MUTEX
                     0213  1346          ENBINT                          ;RESTORE IPL TO ITS VALUE ON ENTRY
50     01    3C      0216  1347          MOVZWL  #SS$_NORMAL,R0          ;SUCCESS ALWAYS
                     05    0219  1348          RSB                             ;RETURN STATUS
                           021A  1349
                           021A  1350 .        .PAGE
```

```
                     021A  1352          .SBTTL  LNM$DELETE_TAB  - DELETE ALL ENTRIES IN A LOGICAL NAME TABLE
                     021A  1353 ;+
                     021A  1354 ; LNM$DELETE_TAB - DELETE ALL ENTRIES IN A LOGICAL NAME TABLE
                     021A  1355 ;
                     021A  1356 ; THIS ROUTINE IS CALLED TO DELETE ALL NAMES WITH A SPECIFIED OR GREATER
                     021A  1357 ; ACCESS MODE FROM A SPECIFIED LOGICAL NAME TABLE.
                     021A  1358 ;
                     021A  1359 ; INPUTS:
                     021A  1360 ;
                     021A  1361 ;       R1 = ADDRESS OF TABLE HEADER.
                     021A  1362 ;       R2 = ACCESS MODE.
                     021A  1363 ;
                     021A  1364 ;       IT IS ASSUMED THAT THE LOGICAL NAME MUTEX IS LOCKED FOR WRITE ACCESS,
                     021A  1365 ;       AND THAT THE CALLER HAS THE PRIVILEGE OF DELETING THE LOGICAL NAME
                     021A  1366 ;       TABLE ENTRIES WITHIN THE SPECIFIED TABLE.
                     021A  1367 ;
                     021A  1368 ; OUTPUTS:
                     021A  1369 ;
                     021A  1370 ;       THE APPROPRIATE HASH TABLE IS SCANNED AND ALL NAMES CONTAINED IN
                     021A  1371 ;       THE SPECIFIED TABLE WITH AN ACCESS MODE GREATER THAN OR EQUAL TO
                     021A  1372 ;       THE SPECIFIED ACCESS MODE ARE DELETED.  QUOTA IS RETURNED.
                     021A  1373 ;
                     021A  1374 ;       R1, R2, AND R3 ARE MODIFIED.
                     021A  1375 ;-
                     021A  1376
                     021A  1377 LNM$DELETE_TAB::                        ;DELETE LOGICAL NAME TABLE ENTRY
   7E    54    7D    021A  1378          MOVQ    R4,-(SP)              ;SAVE REGISTERS
53    01 A1    D0    021D  1379          MOVL    LNMTH$L_HASH(R1),R3   ;ADDRESS OF HASH TABLE
      FEA0    30    0221  1380          BSBW    DELETE_NAMES          ;DELETE THE NAMES WITHIN THE TABLE
   54    8E    7D    0224  1381          MOVQ    (SP)+,R4              ;RESTORE REGISTERS
   50    01    3C    0227  1382          MOVZWL  #SS$_NORMAL,R0        ;SUCCESS ALWAYS
               05    022A  1383          RSB                           ;RETURN STATUS
                     022B  1384
                     022B  1385 ;       .PAGE
```

```
                                022B  1387          .SBTTL  LNM$INIT_PROT   - INIT A LOGICAL NAME TABLE'S OBJECT RIGHTS BLOCK
                                022B  1388 ;+
                                022B  1389 ; LNM$INIT_PROT - INIT A LOGICAL NAME TABLE'S OBJECT RIGHTS BLOCK
                                022B  1390 ;
                                022B  1391 ; THIS ROUTINE IS CALLED TO INITIALIZE THE OBJECT RIGHTS BLOCK FOR A SHAREABLE
                                022B  1392 ; LOGICAL NAME TABLE. IT IS ASSUMED THAT THIS ROUTINE IS NEVER CALLED FOR A
                                022B  1393 ; PROCESS-PRIVATE LOGICAL NAME TABLE. CURRENTLY, ONLY SOGW PROTECTION AND UIC
                                022B  1394 ; ARE ACTUALLY USED.   ACCESS MODE CHECKING IS HANDLED BY THE CALLING ROUTINE
                                022B  1395 ; AND ALL OTHER SECURITY CHECKS ARE NYI.
                                022B  1396 ;
                                022B  1397 ; INPUTS:
                                022B  1398 ;
                                022B  1399 ;       R0 = ADDRESS OF STORAGE TO USE FOR ORB.
                                022B  1400 ;       R1 = ADDRESS OF PARENT LOGICAL NAME TABLE LNMTH.
                                022B  1401 ;       R2 = ADDRESS OF LOGICAL NAME TABLE LNMTH.
                                022B  1402 ;       R4 = ADDRESS OF LOGICAL NAME TABLE LNMB.
                                022B  1403 ;       R6 = ACCESS MASK TO ASSIGN TO THE TABLE.
                                022B  1404 ;       4(SP) = CURRENT PCB ADDRESS.
                                022B  1405 ;
                                022B  1406 ; OUTPUTS:
                                022B  1407 ;
                                022B  1408 ;       R0, R3 DESTROYED.
                                022B  1409 ;       ALL OTHER REGISTERS PRESERVED.
                                022B  1410 ;-
                                022B  1411
                                022B  1412
                                022B  1413 LNM$INIT_PROT::                              ;INIT ORB FOR A LOGICAL NAME TABLE
          50    07    C0        022B  1414          ADDL2   #^X07,R0                  ;ALGIN ORB ON A QUADWORD BOUNDRY
          50    07    CA        022E  1415          BICL2   #^X07,R0
       05 A2    60    DE        0231  1416          MOVAL   (R0),LNMTH$L_ORB(R2)      ;SAVE ADDRESS OF ORB IN TABLE HEADER
       53    04    AE    D0    0235  1417          MOVL    4(SP),R3                  ;RETRIEVE PCB ADDRESS
    80    00BC    C3    D0    0239  1418          MOVL    PCB$L_UIC(R3),(R0)+       ;SET OWNER UIC
    80    FFFF    8F    3C    023E  1419          MOVZWL  #^XFFFF,(R0)+             ;INITIALIZE ACL MUTEX
80   00490058    8F    D0    0243  1420          MOVL    #<DYN$C_ORB @ 16+-        ;SET STRUCTURE TYPE IN FLAGS FIELD
                                024A  1421                  ORB$C_LENGTH>,(R0)+       ;SET STRUCTURE SIZE
                      80    D4    024A  1422          CLRL    (R0)+                     ;SPARE WORD AND REF COUNT NOT USED
                      80    7C    024C  1423          CLRQ    (R0)+                     ;NO ACCESS MODE CHECKS ARE MADE
80    56    04    00    EF    024E  1424          EXTZV   #0, #4,R6,(R0)+           ;SET SYSTEM PROTECTION FIELD
80    56    04    04    EF    0253  1425          EXTZV   #4, #4,R6,(R0)+           ;SET OWNER PROTECTION FIELD
80    56    04    08    EF    0258  1426          EXTZV   #8, #4,R6,(R0)+           ;SET GROUP PROTECTION FIELD
80    56    04    0C    EF    025D  1427          EXTZV   #12,#4,R6,(R0)+           ;SET WORLD PROTECTION FIELD
                      80    7C    0262  1428          CLRQ    (R0)+                     ;NOTE NO ACL AS YET
                      34    BB    0264  1429          PUSHR   #^M<R2,R4,R5>             ;SAVE REGISTERS (R1 PRESERVED BY MOVC)
       00    61    00    2C    0266  1430          MOVC5   #0,(R1),#0,-              ;INITIALIZE MINIMUM AND MAXIMUM
                60    28          026A  1431                  #<ORB$S_MIN_CLASS+-       ;CLASSIFICATION MASKS TO 0
                                026C  1432                  ORB$S_MAX_CLASS>,(R0)
                      34    BA    026C  1433          POPR    #^M<R2,R4,R5>             ;RESTORE REGISTERS
                      05    026E  1434          RSB                               ;RETURN
                                026F  1435
                                026F  1436 ;       .PAGE
```

```
026F  1438          .SBTTL  LNM$INSLOGTAB   - INSERT IN LOGICAL NAME TABLE BY ADDRESS
026F  1439 ;+
026F  1440 ; LNM$INSLOGTAB - INSERT IN LOGICAL NAME TABLE BY ADDRESS
026F  1441 ;
026F  1442 ; THIS ROUTINE IS CALLED TO INSERT A NEW ENTRY INTO THE LOGICAL NAME TABLE
026F  1443 ; SPECIFIED BY TABLE HEADER ADDRESS. INSERTION IN THE CASE OF AN ENTRY FOR A
026F  1444 ; NEW LOGICAL NAME TABLE INCLUDES THE LINKING OF THE NEW TABLE ENTRY TO PARENT
026F  1445 ; AND SIBLINGS AND ANY REQUIRED QUOTA DEDUCTIONS.
026F  1446 ;
026F  1447 ; IF AN ENTRY (OR ENTRIES) ALREADY EXISTS AT AN OUTER ACCESS MODE AND THE NEW
026F  1448 ; NAME IS UNALIASABLE, THE EXISTING NAME(S) IS (ARE) DELETED.
026F  1449 ;
026F  1450 ; IF AN UNALIASABLE ENTRY ALREADY EXISTS IN AN INNER ACCESS MODE, AN ERROR IS
026F  1451 ; RETURNED.
026F  1452 ;
026F  1453 ; IF AN EQUIVALENT ENTRY ALREADY EXISTS, IT IS DELETED UNLESS CREATE_IF IS
026F  1454 ; SPECIFIED IN WHICH CASE THE NEW ENTRY WILL BE JUST BE MAPPED TO THE EXISTING
026F  1455 ; ENTRY, AND NO NAMES ARE SUPERSEDED.
026F  1456 ;
026F  1457 ; IF THE NEW ENTRY IS FOR A LOGICAL NAME TO BE CONTAINED WITHIN EITHER THE
026F  1458 ; PROCESS OR SYSTEM DIRECTORY LOGICAL NAME TABLE, THEN THE HASH CODE VALUE OF
026F  1459 ; EACH OF THE LOGICAL NAME'S EQUIVALENCE STRINGS IS COMPUTED AND STORED WITHIN
026F  1460 ; THE CORRESPONDING TRANSLATION BLOCK.
026F  1461 ;
026F  1462 ; INPUTS:
026F  1463 ;
026F  1464 ;       R1 = ADDRESS OF LOGICAL NAME BLOCK.
026F  1465 ;            (ADDRESS OF TABLE IS IN THE BLOCK).
026F  1466 ;       R2 = ATTRIBUTES AFFECTING TABLE ENTRY CREATION.
026F  1467 ;            (IT IS ASSUMED THAT THE HIGH ORDER BIT IS UNUSED AND 0).
026F  1468 ;
026F  1469 ;       IT IS ASSUMED THAT THERE IS SUFFICIENT QUOTA IN THE CONTAINING TABLE
026F  1470 ;       (AND THE PARENT'S QUOTA HOLDER IN THE CASE OF A LOGICAL NAME TABLE
026F  1471 ;       ENTRY) FOR THE INSERTION OF THE NEW ENTRY AND THE DEDUCTION OF ITS
026F  1472 ;       SEPARATE QUOTA WHEN APPROPRIATE.
026F  1473 ;
026F  1474 ;       IT IS ASSUMED THAT THE LOGICAL NAME MUTEX IS LOCKED FOR WRITE ACCESS.
026F  1475 ;
026F  1476 ; OUTPUTS:
026F  1477 ;
026F  1478 ;       R0 CONTAINS A SUCCESS STATUS.
026F  1479 ;
026F  1480 ;               THE LOGICAL NAME IS INSERTED IN THE SPECIFIED TABLE.
026F  1481 ;               QUOTA IS DEDUCTED WHEN APPROPRIATE.
026F  1482 ;
026F  1483 ;               R0 = SS$_LNMCREATED - NEW TABLE WAS INSERTED.
026F  1484 ;               R0 = SS$_NORMAL     - NEW TABLE WAS MAPPED TO EXISTING TABLE OR
026F  1485 ;                                     NEW LOGICAL NAME WAS INSERTED.
026F  1486 ;               R0 = SS$_SUPERSEDE  - LOGICAL NAME SUPERCEDED EXISTING NAME.
026F  1487 ;
026F  1488 ;       R1 CONTAINS ADDRESS OF LNMB MAPPED TO IF CREATE-IF SET AND NEW LOGICAL
026F  1489 ;               NAME TABLE ENTRY IS MAPPED TO EXISTING ENTRY.
026F  1490 ;
026F  1491 ;       R0 CONTAINS AN ERROR STATUS.
026F  1492 ;
026F  1493 ;               R0 = SS$_DUPLNAME   - NON-ALIASABLE DUPLICATE EXISTS.
026F  1494 ;               R0 = SS$_PARENT_DEL - DELETEION OF PARENT WOULD HAVE OCCURRED.
```

```
                                026F  1495 ;
                                026F  1496 ;
                                026F  1497 ;       R1, R2, R3, AND R5 ARE MODIFIED
                                026F  1498 ;
                                026F  1499 ;-
                                026F  1500
                                026F  1501 LNM$INSLOGTAB::                          ;INSERT IN LOGICAL NAME TABLE ENTRY
          55    51    D0      026F  1502         MOVL    R1,R5                     ;ADDRESS OF NAME BLOCK
                                0272  1503
                                0272  1504 ;
                                0272  1505 ; BUILD A NAME TRANSLATION CONTROL BLOCK FOR THE NEW TABLE ENTRY ON THE STACK.
                                0272  1506 ;
                                0272  1507
              7E    D4        0272  1508         CLRL    -(SP)                     ;ZERO NT_L_THREAD
            0C A5   DD        0274  1509         PUSHL   LNMB$L_TABLE(R5)          ;ADDRESS OF CONTAINING TABLE HEADER
            12 A5   9F        0277  1510         PUSHAB  LNMB$T_NAME+1(R5)         ;ADDRESS OF NAME
       7E   11 A5   9A        027A  1511         MOVZBL  LNMB$T_NAME(R5),-(SP)     ;LENGTH OF NAME
    00000403 8F     DD        027E  1512         PUSHL   #<NT_M_MODIFY @ 8+-       ;HASH BUCKET MIGHT BE MODIFIED
                              0284  1513                 PSL$C_USER>               ;SPECIFY ACCESS MODE AS USER
 04 10 A5     00    E0        0284  1514         BBS     #LNMB$V_NO_ALIAS, -       ;BRANCH IF NOT ALIASABLE AND ALL ACCESS
                              0289  1515                 LNMB$B_FLAGS(R5),10$      ;MODES WILL BE CONSIDERED
       6E    0B A5   90        0289  1516         MOVB    LNMB$B_ACMODE(R5),(SP)    ;SKIP CONSIDERATION OF OUTER ACCESS MODE
          53    5E    D0      028D  1517 10$:     MOVL    SP,R3                     ;ADDRESS OF TRANSLATION BLOCK
                              0290  1518
                              0290  1519 ;
                              0290  1520 ; SEARCH FOR AN EXISTING LOGICAL NAME TABLE ENTRY WITH A NAME MATCHING THE NAME
                              0290  1521 ; OF THE NEW TABLE ENTRY. THERE CAN BE TEN OUTCOMES OF SUCH A SEARCH AND THEY
                              0290  1522 ; ARE LISTED BELOW:
                              0290  1523 ;
                              0290  1524 ; 1. AN EXISTING LOGICAL NAME TABLE ENTRY WAS NOT FOUND AT THE ACCESS MODE
                              0290  1525 ;    SEARCHED.
                              0290  1526 ;
                              0290  1527 ;       A. THE CREATE_IF BIT IS SET:
                              0290  1528 ;          IF IDENTICAL NAMES AT OUTER ACCESS MODE HAD BEEN SEEN, THEN THESE
                              0290  1529 ;          LOGICAL NAME TABLE ENTRIES MUST BE DELETED BEFORE THE NEW ENTRY CAN BE
                              0290  1530 ;          INSERTED. THIS IS DONE BY RE-STARTING FROM THE BEGINNING THE SEARCH FOR
                              0290  1531 ;          AN EXISTING LOGICAL NAME TABLE ENTRY WITH A NAME MATCHING THE NAME OF
                              0290  1532 ;          THE NEW TABLE ENTRY AFTER CLEARING THE CREATE_IF BIT.
                              0290  1533 ;
                              0290  1534 ;       B. THE CREATE_IF BIT IS NOT SET:
                              0290  1535 ;          THE NEW LOGICAL NAME TABLE ENTRY IS INSERTED.
                              0290  1536 ;
                              0290  1537 ; 2. AN EXISTING LOGICAL NAME TABLE ENTRY IS FOUND BUT IT IS AT AN INNER ACCESS
                              0290  1538 ;    MODE.
                              0290  1539 ;
                              0290  1540 ;       A. THE EXISTING LOGICAL NAME TABLE ENTRY DOES NOT ALLOW ALLIASES:
                              0290  1541 ;          AN ERROR IS RETURNED.
                              0290  1542 ;
                              0290  1543 ;       B. THE CREATE_IF BIT IS SET AND THE EXISTING NAME ALLOWS ALLIASES:
                              0290  1544 ;          IF IDENTICAL NAMES AT OUTER ACCESS MODE HAD BEEN SEEN, THEN THESE
                              0290  1545 ;          LOGICAL NAME TABLE ENTRIES MUST BE DELETED BEFORE THE NEW ENTRY CAN BE
                              0290  1546 ;          INSERTED. THIS IS DONE BY RE-STARTING FROM THE BEGINNING THE SEARCH FOR
                              0290  1547 ;          AN EXISTING LOGICAL NAME TABLE ENTRY WITH A NAME MATCHING THE NAME OF
                              0290  1548 ;          THE NEW TABLE AFTER CLEARING THE CREATE_IF BIT.
                              0290  1549 ;
                              0290  1550 ;       C. THE CREATE_IF BIT IS NOT SET AND THE EXISTING NAME ALLOWS ALLIASES:
                              0290  1551 ;          THE NEW LOGICAL NAME TABLE ENTRY IS INSERTED.
```

```
                     0290  1552 ;
                     0290  1553 ; 3. AN EXISTING LOGICAL NAME TABLE ENTRY IS FOUND BUT IT IS AT AN OUTER ACCESS
                     0290  1554 ;    MODE.
                     0290  1555 ;
                     0290  1556 ;       A. THE CREATE_IF BIT IS SET:
                     029C  1557 ;          A RECORD IS KEPT THAT A LOGICAL NAME TABLE ENTRY WITH THE SAME NAME AS
                     0290  1558 ;          THE NEW TABLE ENTRY WAS SEEN, AND THE SEARCH CONTINUES.
                     0290  1559 ;
                     0290  1560 ;       B. THE CREATE_IF BIT IS NOT SET:
                     0290  1561 ;          THE EXISTING LOGICAL NAME TABLE ENTRY IS DELETED, AND THE SEARCH
                     0290  1562 ;          CONTINUES.
                     0290  1563 ;
                     0290  1564 ; 4. AN EXISTING LOGICAL NAME TABLE ENTRY IS FOUND AT THE SAME ACCESS MODE.
                     0290  1565 ;
                     0290  1566 ;       A. THE CREATE_IF BIT IS SET:
                     0290  1567 ;          THE NEW ENTRY IS MAPPED TO THE EXISTING ENTRY WITH THE SAME NAME AND
                     0290  1568 ;          ACCESS MODE.
                     0290  1569 ;
                     0290  1570 ;       B. THE CREATE_IF BIT IS NOT SET BUT THE NODELETE BIT IS:
                     0290  1571 ;          AN ERROR IS RETURNED INFORMING THAT THE CALLER ATTEMPTED TO DELETE
                     0290  1572 ;          THE PARENT TABLE SPECIFIED (EITHER DIRECTLY OR INDIRECTLY THROUGH THE
                     0290  1573 ;          DELETION OF A GRANDPARENT).
                     0290  1574 ;
                     0290  1575 ;       C. NEITHER THE CREATE_IF BIT NOR THE NODELETE BIT IS SET:
                     0290  1576 ;          THE EXISTING ENTRY IS DELETED, AND THE SEARCH CONTINUES.
                     0290  1577 ;
                     0290  1578 ;
                     0290  1579
            52   DD  0290  1580          PUSHL   R2                      ;SAVE THE ATTRIBUTES
            01   DD  0292  1581          PUSHL   #SS$_NORMAL             ;ASSUME SUCCESS
          0234   30  0294  1582 20$:     BSBW    LNM$PRESEARCH           ;SEARCH FOR NAME IN TABLE
                     0297  1583
        4C 50    E9  0297  1584 25$:     BLBC    R0,50$                  ;NOTHING FOUND
OB A5   0B A1    91  029A  1585          CMPB    LNMB$B_ACMODE(R1), -    ;COMPARE ACCESS MODES
                     029F  1586                  LNMB$B_ACMODE(R5)
            3D   1F  029F  1587          BLSSU   40$                     ;BRANCH IF OWNED BY AN INNER MODE
            10   1A  02A1  1588          BGTRU   30$                     ;BRANCH IF OWNED BY AN OUTER MODE
                     02A3  1589
                     02A3  1590 ;
                     02A3  1591 ; AN EXISTING LOGICAL NAME TABLE ENTRY AT THE SAME ACCESS MODE WAS FOUND TO
                     02A3  1592 ; HAVE THE SAME NAME AS THAT OF THE NEW TABLE ENTRY. EITHER MAP THE NEW ENTRY
                     02A3  1593 ; TO THE EXISTING ENTRY, OR DELETE THE EXISTING ENTRY DEPENDING UPON THE
                     02A3  1594 ; SETTING OF CREATE_IF.
                     02A3  1595 ;
                     02A3  1596
03 04 AE    18   E1  02A3  1597          BBC     #LNM$V_CREATE_IF,4(SP),27$ ;GO RETURN IF MAPPING NEW TABLE ENTRY
          00E5   31  02A8  1598          BRW     90$                     ;ONTO EXISTING TABLE ENTRY
            04   E1  02AB  1599 27$:     BBC     #LNMB$V_NODELETE,-      ;IF TABLE ENTRY IS A (GRAND)PARENT TO
       03 10 A1      02AD  1600                  LNMB$B_FLAGS(R1),30$    ;THE NEW TABLE ENTRY THEN RETURN ERROR
          00EB   31  02B0  1601          BRW     110$
                     02B3  1602
                     02B3  1603 ;
                     02B3  1604 ; AN EXISTING LOGICAL NAME TABLE ENTRY AT AN OUTER ACCESS MODE WAS FOUND TO
                     02B3  1605 ; HAVE THE SAME NAME AS THAT OF THE NEW TABLE ENTRY. EITHER REMEMBER THAT SUCH
                     02B3  1606 ; A LOGICAL NAME TABLE ENTRY HAS BEEN ENCOUNTERED, OR DELETE IT DEPENDING UPON
                     02B3  1607 ; THE SETTING OF CREATE_IF.
                     02B3  1608 ;
```

```
                                                G 16
                     02B3  1609
0B 04 AE    18   E1  02B3  1610 30$:    BBC     #LNM$V_CREATE_IF,4(SP),35$ ;BRANCH IF CREATE_IF NOT SET
00 04 AE    1F   E2  02B8  1611         BBSS    #31,4(SP),31$              ;MARK OUTER ACCESS NAMES SEEN
   10 A3    61   9E  02BD  1612 31$:    MOVAB   (R1),NT_L_THREAD(R3)       ;SETUP TO CONTINUE WITH NEXT BLOCK
            16   11  02C1  1613         BRB     37$                        ;AND GO CONTINUE SEARCH
                     02C3  1614
 6E   0631  8F   3C  02C3  1615 35$:    MOVZWL  #SS$_SUPERSEDE,(SP)        ;UPDATE STATUS RETURN
            53   DD  02C8  1616         PUSHL   R3                         ;SAVE REGISTERS OVER DELETE
          FED9   30  02CA  1617         BSBW    LNM$DELETE                 ;DELETE A LOGICAL NAME BLOCK
            53 8ED0  02CD  1618         POPL    R3                         ;RESTORE THINGS
         06 50   E8  02D0  1619         BLBS    R0,37$                     ;CONTINUE IF DELETION SUCCESSFUL
       6E   50   D0  02D3  1620         MOVL    R0,(SP)                    ;OTHERWISE SAVE ERROR AND TERMINATE
          00B7   31  02D6  1621         BRW     90$                        ;TABLE ENTRY INSERTION ATTEMPT
                     02D9  1622
          0219   30  02D9  1623 37$:    BSBW    LNM$CONTSEARCH             ;CONTINUE SEARCHING FOR MORE MODES
            B9   11  02DC  1624         BRB     25$                        ;LOOP OVER ALL ALIASING NAMES
                     02DE  1625
                     02DE  1626 ;
                     02DE  1627 ; AN EXISTING LOGICAL NAME TABLE ENTRY AT AN INNER ACCESS MODE WAS FOUND TO
                     02DE  1628 ; HAVE THE SAME NAME AS THAT OF THE NEW TABLE ENTRY. IF THIS LOGICAL NAME TABLE
                     02DE  1629 ; ENTRY DOES NOT ALLOW ALLIASES THEN RETURN AN ERROR. OTHERWISE, EITHER INSERT
                     02DE  1630 ; THE NEW TABLE ENTRY, OR RE-START THE SEARCH FROM THE BEGINNING DEPENDING UPON
                     02DE  1631 ; WHETHER CREATE_IF IS SET, AND IF IT IS, WHETHER ANY IDENTICAL NAMES WERE
                     02DE  1632 ; FOUND TO EXIST AT OUTER ACCESS MODES.
                     02DE  1633 ;
                     02DE  1634
03 10 A1    00   E1  02DE  1635 40$:    BBC     #LNMB$V_NO_ALIAS, -        ;CONTINUE IF ALIASABLE
                     02E3  1636                 LNMB$B_FLAGS(R1),50$
          00B1   31  02E3  1637         BRW     100$                       ;BRANCH IF NOT ALIASABLE
                     02E6  1638
                     02E6  1639 ;
                     02E6  1640 ; AN EXISTING LOGICAL NAME TAB E ENTRY WAS NOT FOUND TO HAVE THE SAME SAME NAME
                     02E6  1641 ; AS THAT OF THE NEW TABLE ENTRY AT THE ACCESS MODE SEARCHED. EITHER INSERT THE
                     02E6  1642 ; NEW TABLE ENTRY, OR RE-START THE SEARCH FROM THE BEGINNING DEPENDING UPON
                     02E6  1643 ; WHETHER CREATE_IF IS SET, AND IF IT IS, WHETHER ANY IDENTICAL NAMES WERE
                     02E6  1644 ; FOUND TO EXIST AT OUTER ACCESS MODES.
                     02E6  1645 ;
                     02E6  1646
0A 04 AE    18   E5  02E6  1647 50$:    BBCC    #LNM$V_CREATE_IF,4(SP),55$;INSERT NEW ENTRY IF CREATE_IF NOT SET
05 04 AE    1F   E5  02EB  1648         BBCC    #31,4(SP),55$              ;OR IF OUTER ACCESS MODE NAMES NOT SEEN
         10 A3   D4  02F0  1649         CLRL    NT_L_THREAD(R3)            ;GET SET TO RE-START SEARCH
            9F   11  02F3  1650         BRB     20$                        ;GO RE-START SEARCH
                     02F5  1651
   52    10 A3   D0  02F5  1652 55$:    MOVL    NT_L_THREAD(R3),R2         ;ADDRESS OF PREVIOUS LNMB BLOCK
   04 A5    52   D0  02F9  1653         MOVL    R2,LNMB$L_BLINK(R5)        ;SET NEW LNMB BLOCKS BACKPOINTER
      62    55   D0  02FD  1654         MOVL    R5,LNMB$L_FLINK(R2)        ;RESET PREVIOUS LNMB BLOCKS FRONTPOINTER
      65    51   D0  0300  1655         MOVL    R1,LNMB$L_FLINK(R5)        ;SET NEW LNMB BLOCKS FRONTPOINTER
            04   13  0303  1656         BEQL    60$                        ;BRANCH IF NEW LNMB BLOCK IS AT LIST END
   04 A1    55   D0  0305  1657         MOVL    R5,LNMB$L_BLINK(R1)        ;RESET NEXT LNMB BLOCKS BACKPOINTER
                     0309  1658
                     0309  1659 ;
                     0309  1660 ; IF THE NEW ENTRY IS A LOGICAL NAME TABLE THEN THE FOLLOWING ACTIONS ARE
                     0309  1661 ; TAKEN:
                     0309  1662 ;
                     0309  1663 ; 1. THE LOGICAL NAME BLOCK IS LINKED IN AS THE IMMEDIATE CHILD OF ITS PARENT
                     0309  1664 ;    TABLE, AND THUS, AS THE FIRST SIBLING IN THE LIST OF SIBLINGS.
                     0309  1665 ;
```

```
                          0309  1666 ; 2. THE STATUS OF THE INSERTION IS CHANGED TO SS$_LNMCREATED IF NO LOGICAL NAME
                          0309  1667 ;    TABLE ENTRIES HAD BEEN DELETED.
                          0309  1668 ;
                          0309  1669 ; 3. QUOTA CONSISTING OF ANY QUOTA SPECIFICALLY ALLOCATED TO THE NEW TABLE IS
                          0309  1670 ;    DEDUCTED FROM THE QUOTA HOLDER OF THE PARENT LOGICAL NAME TABLE.
                          0309  1671 ;
                          0309  1672
                    03    E1    0309  1673 60$:    BBC     #LNMB$V_TABLE,-          ;BRANCH IF THE NAME BLOCK IS NOT FOR
                2F 10 A5          030B  1674                 LNMB$B_FLAGS(R5),80$     ;A NEW LOGICAL NAME TABLE
                                  030E  1675
              50  11 A5    9A    030E  1676 70$:    MOVZBL  LNMB$T_NAME(R5),R0       ;SIZE OF NAME STRING
            52  12 A540    9E    0312  1677         MOVAB   LNMB$T_NAME+1(R5)[R0],R2;ADDRESS OF TRANSLATION BLOCK
              52  05 A2    9E    0317  1678         MOVAB   LNMX$T_XLATION+1(R2),R2 ;ADDRESS OF BLOCKS TABLE HEADER
                                  031B  1679
              51  0D A2    D0    031B  1680         MOVL    LNMTH$L_PARENT(R2),R1    ;ADDRESS OF PARENT'S TABLE HEADER
                  11 A1    D0    031F  1681         MOVL    LNMTH$L_CHILD(R1),-      ;LINK IN NEW TABLE ENTRY AS THE
                  15 A2          0322  1682                 LNMTH$L_SIBLING(R2)      ;IMMEDIATE CHILD OF THE PARENT AND
              11 A1   62    9E    0324  1683         MOVAB   (R2),LNMTH$L_CHILD(R1)   ;AS THE FIRST SIBLING IN THE LIST
                                  0328  1684
            6E  0631 8F     B1    0328  1685         CMPW    #SS$_SUPERSEDE,(SP)      ;WAS A LOGIGAL NAME SUPERSEDED?
                    05      13    032D  1686         BEQL    75$                      ;BRANCH IF YES
            6E  06B1 8F     3C    032F  1687         MOVZWL  #SS$_LNMCREATED,(SP)     ;CHANGE STATUS IF NO
                                  0334  1688
              51  19 A1    D0    0334  1689 75$:    MOVL    LNMTH$L_QTABLE(R1),R1    ;RETRIEVE PARENT'S QUOTA HOLDER
                  1D A2    C2    0338  1690         SUBL2   LNMTH$L_BYTESLM(R2),-    ;SUBTRACT QUOTA TO BE SPECIFICALLY
                  21 A1          033B  1691                 LNMTH$L_BYTES(R1)        ;ALLOCATED TO THE NEW TABLE
                                  033D  1692
                                  033D  1693 ;
                                  033D  1694 ; DEDUCT THE SIZE OF THE NEW LOGICAL NAME ENTRY FROM THE QUOTA HOLDER OF THE
                                  033D  1695 ; CONTAINING LOGICAL NAME TABLE.
                                  033D  1696 ;
                                  033D  1697
            53  08 A5    3C    033D  1698 80$:    MOVZWL  LNMB$W_SIZE(R5),R3       ;RETRIEVE SIZE OF NEW LOGICAL NAME ENTRY
            51  0C A5    D0    0341  1699         MOVL    LNMB$L_TABLE(R5),R1      ;RETRIEVE CONTAINING TABLE HEADER ADDR
            52  19 A1    D0    0345  1700         MOVL    LNMTH$L_QTABLE(R1),R2    ;RETRIEVE QUOTA HOLDER'S ADDRESS
          21 A2   53      C2    0349  1701         SUBL2   R3,LNMTH$L_BYTES(R2)     ;SUBTRACT SIZE OF NEW TABLE ENTRY
                                  034D  1702
                                  034D  1703 ;
                                  034D  1704 ; BUMP THE APPROPRIATE DIRECTORY SEQUENCE NUMBER IF THE CONTAINING TABLE
                                  034D  1705 ; IS ONE OF THE DIRECTORY TABLES.
                                  034D  1706 ;
                                  034D  1707
                    01    E1    034D  1708         BBC     #LNMTH$V_DIRECTORY,-     ;DIRECTORY TABLE?
                3F 61           034F  1709                 LNMTH$B_FLAGS(R1),90$    ;OKAY IF NOT
            08 51   1F    E0    0351  1710         BBS     #31,R1,82$               ;BRANCH IF SYSTEM DIRECTORY
        00000000'9F       D6    0355  1711         INCL    @#CTL$GL_LNMDIRSEQ       ;BUMP PROCESS DIRECTORY COUNTER
                    06    11    035B  1712         BRB     83$
        00000000'9F       D6    035D  1713 82$:    INCL    @#LNM$GL_SYSDIRSEQ       ;BUMP SYSTEM DIRECTORY COUNTER
                                  0363  1714
                                  0363  1715 ;
                                  0363  1716 ; IF THE NEW ENTRY IS FOR A LOGICAL NAME TO BE CONTAINED WITHIN EITHER THE
                                  0363  1717 ; PROCESS OR SYSTEM DIRECTORY LOGICAL NAME TABLE, THEN THE HASH CODE VALUE OF
                                  0363  1718 ; EACH OF THE LOGICAL NAME'S EQUIVALENCE STRINGS IS COMPUTED AND STORED WITHIN
                                  0363  1719 ; THE CORRESPONDING TRANSLATION BLOCK.
                                  0363  1720 ;
                                  0363  1721
                    03    E0    0363  1722 83$:    BBS     #LNMB$V_TABLE,-          ;SKIP COMPUTATION AND STORAGE OF HASH
```

```
                                                 I 16
                     28 10 A5          0365  1723                LNMB$B_FLAGS(R5),90$    ;CODES IF THIS IS A LOGICAL NAME TABLE
                                       0368  1724
                  55   11 A5    9E    0368  1725        MOVAB   LNMB$T_NAME(R5),R5       ;RETRIEVE ADDRESS AND SIZE OF LOGICAL
                       50   85    9A    036C  1726        MOVZBL  (R5)+,R0                ;NAME'S NAME
                       55   50    C0    036F  1727        ADDL2   R0,R5                   ;POSITION TO FIRST TRANSLATION BLOCK
                                       0372  1728
                            02    E0    0372  1729 85$:    BBS     #LNMX$V_XEND,-          ;GO RETURN IF LAST TRANSLATION BLOCK
                          1A 65         0374  1730                LNMX$B_FLAGS(R5),90$
                  51   04 A5    9E    0376  1731        MOVAB   LNMX$T_XLATION(R5),R1   ;RETRIEVE ADDRESS AND SIZE OF CURRENT
                       50   81    9A    037A  1732        MOVZBL  (R1)+,R0                ;TRANSLATION BLOCK'S EQUIVALENCE STRING
                       7E   50    7D    037D  1733        MOVQ    R0,-(SP)                ;SAVE ADDRESS AND SIZE ON STACK
                         01EB    30    0380  1734        BSBW    LNM$HASH                ;DETERMINE AND STORE THE CURRENT
                02 A5  50    B0    0383  1735        MOVW    R0,LNMX$W_HASH(R5)      ;EQUIVALENCE STRING'S HASH CODE VALUE
                       50   8E    7D    0387  1736        MOVQ    (SP)+,R0                ;RESTORE ADDRESS AND SIZE OF STRING
                55     51   50    C1    038A  1737        ADDL3   R0,R1,R5                ;POSITION TO NEXT TRANSLATION BLOCK
                            E2    11    038E  1738        BRB     85$                     ;AND CONTINUE
                                       0390  1739
                                       0390  1740 ;
                                       0390  1741 ; RETRIEVE THE STATUS TO BE RETURNED, CLEAN THE NAME TRANSLATION CONTROL BLOCK
                                       0390  1742 ; FROM THE STACK AND RETURN.
                                       0390  1743 ;
                                       0390  1744
                            50 8ED0    0390  1745 90$:    POPL    R0                      ;FETCH STATUS
                       5E   18    C0    0393  1746        ADDL    #NT_K_LENGTH+4,SP       ;CLEAN BLOCK FROM STACK
                                  05    0396  1747        RSB                             ;EXIT
                6E   0094 8F    3C    0397  1748 100$:   MOVZWL  #SS$_DUPLNAM,(SP)       ;TRIED TO SUPERCEDE UNALIASABLE NAME
                            F2    11    039C  1749        BRB     90$                     ;JOIN MAIN EXIT
                6E   2254 8F    3C    039E  1750 110$:   MOVZWL  #SS$_PARENT_DEL,(SP)    ;TRIED TO DELETE (GRAND)PARENT
                            EB    11    03A3  1751        BRB     90$                     ;JOIN MAIN EXIT
                                       03A5  1752
                                       03A5  1753 ;       .PAGE
```

```
                                                J 16
                                                
                                                
                                                
```

```
                03 50   E8  03C8  1812          BLBS    R0,10$                          ;BRANCH IF NO POSSIBLE TRANSLATION
                10 A3   D4  03CB  1813          CLRL    NT_L_THREAD(R3)                 ;CLEAR NAME BLOCK ADDRESS
                            03CE  1814
                   7E   7C  03CE  1815 10$:     CLRQ    -(SP)                           ;MAKE ANOTHER NAME TRANSLATION BLOCK
             7E    04 A3   7D  03D0  1816          MOVQ    NT_L_NAMLEN(R3),-(SP)           ;TABLE NAME AND ADDRESS
                   63      DD  03D4  1817          PUSHL   (R3)                            ;CONTROL LONGWORD
          53       5E      D0  03D6  1818          MOVL    SP,R3                           ;ADDRESS OF SYSTEM NAME TRANSLATION BLOCK
          51       95      D0  03D9  1819          MOVL    @(R5)+,R1                       ;ADDRESS OF SYSTEM HASH TABLE
                            03DC  1820
                 00EC      30  03DC  1821          BSBW    LNM$PRESEARCH                   ;FIND HEAD OF POSSIBLE TRANSLATIONS
                08 50      E8  03DF  1822          BLBS    R0,20$                          ;BRANCH IF TRANSLATION MAY EXIST
                10 A3      D4  03E2  1823          CLRL    NT_L_THREAD(R3)                 ;NO POSSIBLE TRANSLATION
                            03E5  1824
                            03E5  1825 ;
                            03E5  1826 ; IF THE SPECIFIED LOGICAL NAME DOES NOT EXIST IN EITHER THE PROCESS-PRIVATE OR
                            03E5  1827 ; SHAREABLE NAME SPACES, REGARDLESS OF CONTAINING TABLE HEADER ADDRESS OR ACCESS
                            03E5  1828 ; MODE, THEN IT IS POINTLESS TO CONTINUE THE SEARCH, SO RETURN AN ERROR.
                            03E5  1829 ; OTHERWISE, POSITION TO THE FIRST TARGET LOGICAL NAME TABLE IN WHICH TO SEARCH
                            03E5  1830 ; FOR THE SPECIFIED LOGICAL NAME.
                            03E5  1831 ;
                            03E5  1832
                24 A3      D5  03E5  1833          TSTL    NT_K_LENGTH+NT_L_THREAD(R3)     ;ANY POSSIBLE TRANSLATION?
                   44      13  03E8  1834          BEQL    90$                             ;BRANCH IF NO TRANSLATION POSSIBLE
                            03EA  1835 20$:
             5E    2C      C2  03EA  1836          SUBL    #RT_K_LENGTH-4,SP               ;ALLOCATE RECURSIVE TABLE NAME CONTROL BLOCK
                   63      DD  03ED  1837          PUSHL   NT_B_ACMODE(R3)                 ;ACMODE, CASE FLAG
          55       5E      D0  03EF  1838          MOVL    SP,R5                           ;ADDRESS OF BLOCK
       52    28 A3      7D  03F2  1839          MOVQ    2*NT_K_LENGTH(R3),R2            ;GET LOGICAL NAME TABLE DESCRIPTOR
                            03F6  1840
                 01F7      30  03F6  1841          BSBW    LNM$SETUP                       ;SETUP TABLE PROCESSING
                27 50      E9  03F9  1842          BLBC    R0,70$                          ;NO TABLE FOUND
                            03FC  1843
                            03FC  1844 ;
                            03FC  1845 ; SEARCH FOR THE SPECIFIED LOGICAL NAME WITHIN THE CURRENT LOGICAL NAME TABLE.
                            03FC  1846 ; ONLY ONE NAME SPACE WILL HAVE TO BE SEARCHED FOR THE LOGICAL NAME, AND THAT
                            03FC  1847 ; IS THE NAME SPACE THAT CORRESPONDS TO THE NAME SPACE THE CURRENT TABLE RESIDES
                            03FC  1848 ; IN.
                            03FC  1849 ;
                            03FC  1850
       53    30 A5      9E  03FC  1851 40$:     MOVAB   RT_K_LENGTH(R5),R3              ;SYSTEM TABLE CONTROL BLOCK
       03 51    1F      E0  0400  1852          BBS     #3T,R1,50$                      ;BRANCH IF SYSTEM SPACE TABLE
          53    14      C0  0404  1853          ADDL    #NT_K_LENGTH,R3                 ;ADVANCE TO PROCESS TABLE CONTROL BLOCK
                10 A3      D5  0407  1854 50$:     TSTL    NT_L_THREAD(R3)                 ;ANY NAMES TO BE LOOKED AT?
                   11      13  040A  1855          BEQL    60$                             ;NO  - GO GET NEXT TABLE TO PROCESS
       0C A3       51      D0  040C  1856          MOVL    R1,NT_L_TABID(R3)               ;TABLE HEADER ADDRESS
                            0410  1857
                10 A3      DD  0410  1858          PUSHL   NT_L_THREAD(R3)                 ;SAVE SEARCH CONTEXT
                 00DF      30  0413  1859          BSBW    LNM$CONTSEARCH                  ;RESUME SEARCH FOR NAME
       10 A3       8E      D0  0416  1860          MOVL    (SP)+,NT_L_THREAD(R3)           ;RESTORE ORIGINAL SEARCH CONTEXT
                06 50      E8  041A  1861          BLBS    R0,70$                          ;BRANCH IF NAME FOUND
                            041D  1862
                            041D  1863 ;
                            041D  1864 ; THE SPECIFIED LOGICAL NAME WAS NOT FOUND WITHIN THE CURRENT LOGICAL NAME
                            041D  1865 ; TABLE. POSITION TO THE NEXT TABLE IN THE SEARCH LIST OF LOGICAL NAME TABLES.
                            041D  1866 ; IF THERE IS A NEXT TABLE, SEARCH FOR THE LOGICAL NAME WITHIN IT; OTHERWISE,
                            041D  1867 ; THE SEARCH FOR THE SPECIFIED LOGICAL NAME IS TERMINATED WITH AN ERROR.
                            041D  1868 ;
```

```
                                041D  1869
                     01F6   30  041D  1870 60$:    BSBW    LNM$TABLE               ;CONTINUE TABLE PROCESSING
                   D9 50    E8  0420  1871          BLBS    R0,40$                  ;BRANCH TO PROCESS ANOTHER TABLE
                                0423  1872
                5E    58 AE   9E  0423  1873 70$:    MOVAB   <2*NT_K_LENGTH>+RT_K_LENGTH(SP),SP      ;REMOVE TABLE SEARCH CONTROL
                   52    8E   7D  0427  1874 80$:    MOVQ    (SP)+,R2                ;RESTORE REGISTERS
                         55 8ED0  042A  1875          POPL    R5
                            05    042D  1876          RSB
             50    01BC 8F   3C  042E  1877 90$:    MOVZWL  #SS$_NOLOGNAM,R0        ;NO TRANSLATION FOR NAME
                   5E    28  C0  0433  1878          ADDL2   #2*NT_K_LENGTH,SP       ;REMOVE NAME SEARCH CONTROL BLOCK
                         EF  11  0436  1879          BRB     80$                     ;JOIN MAIN EXIT
                                0438  1880 ;        .PAGE
```

```
                     0438  1882          .SBTTL  LNM$SEARCH_ONE  - SEARCH FOR LOGICAL NAME AND RETURN TRANSLATION
                     0438  1883 ;+
                     0438  1884 ; LNM$SEARCH_ONE - SEARCH FOR LOGICAL NAME AND RETURN INDEX 0 TRANSLATION
                     0438  1885 ;
                     0438  1886 ; THIS ROUTINE IS CALLED TO SEARCH FOR A LOGICAL NAME MATCH IN A LIST OF
                     0438  1887 ; LOGICAL NAME TABLES. IF IT FINDS ONE, AND IF THAT LOGICAL NAME HAS A
                     0438  1888 ; TRANSLATION WITH INDEX 0, THEN A COPY OF THE LNMX TRANSLATION BLOCK IS
                     0438  1889 ; RETURNED IN THE SPECIFIED OUTPUT BUFFER.
                     0438  1890 ;
                     0438  1891 ; THIS ROUTINE DOES NOT PERFORM ANY ARGUEMENT VERIFICATION. HOWEVER, THIS
                     0438  1892 ; ROUTINE DOES PERFORM ALL REQUIRED MUTEX LOCKING AND PROTECTION CHECKING.
                     0438  1893 ;
                     0438  1894 ; INPUTS:
                     0438  1895 ;
                     0438  1896 ;       R0 = LENGTH OF LOGICAL NAME STRING.
                     0438  1897 ;       R1 = ADDRESS OF LOGICAL NAME STRING.
                     0438  1898 ;       R2 = LENGTH OF TABLE NAME STRING.
                     0438  1899 ;       R3 = ADDRESS OF TABLE NAME STRING.
                     0438  1900 ;       R4 = PCB ADDRESS
                     0438  1901 ;       R5 = SEARCH ACCESS MODE IN LOW BYTE, CASELESS FLAG IN BIT 8.
                     0438  1902 ;       R6 = ADDRESS OF OUTPUT BUFFER
                     0438  1903 ;            (MUST BE LNM$C_NAMLENGTH + LNMX$T_XLATION BYTES IN SIZE).
                     0438  1904 ;
                     0438  1905 ;
                     0438  1906 ; OUTPUTS:
                     0438  1907 ;
                     0438  1908 ;       R0 LOW BIT CLEAR INDICATES SEARCH FAILURE.
                     0438  1909 ;
                     0438  1910 ;               R0 = SS$_NOLOGNAM - NO LOGICAL NAME MATCH FOUND.
                     0438  1911 ;                                 - LOGICAL NAME FOUND BUT TRANSLATION WITH
                     0438  1912 ;                                   INDEX 0 DOES NOT EXIST.
                     0438  1913 ;
                     0438  1914 ;               R0 = SS$_NOPRIV   - LOGICAL NAME WAS FOUND BUT CALLER DOES NOT
                     0438  1915 ;                                   HAVE ACCESS TO THE SPECIFIED TABLE.
                     0438  1916 ;
                     0438  1917 ;       R0 LOW BIT SET INDICATES SUCCESS WITH:
                     0438  1918 ;
                     0438  1919 ;               A COPY OF THE LNMX FOR TRANSLATION INDEX 0 IN THE OUTPUT BUFFER.
                     0438  1920 ;
                     0438  1921 ;       REGISTERS R1 - R3 AND R5 ARE DESTROYED.
                     0438  1922 ;-
                     0438  1923
                     0438  1924 LNM$SEARCH_ONE::                          ;SEARCH FOR LOGICAL NAME AND RETURN LNMX
           01    DD  0438  1925          PUSHL   #SS$_NORMAL               ;ASSUME SUCCESS
                     043A  1926          SAVIPL                            ;SAVE CURRENT IPL ON STACK
           50    DD  043D  1927          PUSHL   R0                        ;SAVE LOGNAM STRING LENGTH
         0328    30  043F  1928          BSBW    LNM$LOCKR                 ;LOCK LOGICAL NAME MUTEX FOR WRITING
           50 8ED0   0442  1929          POPL    R0                        ;RESTORE LOGNAM STRING LENGTH
                     0445  1930
     00000002      0445  1931          .IF NE CA$_MEASURE                ;CHECK FOR MEASUREMENT ENABLED
00000000'EF    D6  0445  1932          INCL    L^PMS$GL_LOGNAM           ;IF YES COUNT CURRENT TRANSLATION
                     044B  1933          .ENDC
                     044B  1934
         FF57    30  044B  1935          BSBW    LNM$SEARCHLOG             ;SEARCH FOR THE LOGICAL NAME
       58 50    E9  044E  1936          BLBC    R0,40$                    ;EXIT ON ANY ERROR
                     0451  1937
     12 51    1F  E1  0451  1938          BBC     #31,R1,1$                 ;ONLY CHECK ACCESS TO SHAREABLE TABLES
```

```
                                                     B 1
                51    DD  0455  1939          PUSHL   R1                      ;SAVE LNMB ADDRESS
          51    0C A1   DO  0457  1940          MOVL    LNMB$L_TABLE(R1),R1     ;RETRIEVE TABLE HEADER ADDRESS
             52    01   DO  045B  1941          MOVL    #ARM$M_READ,R2          ;READ ACCESS
                FCD0   30  045E  1942          BSBW    LNM$CHECK_PROT          ;PERFORM PROTECTION CHECK
                   51 8ED0  0461  1943          POPL    R1                      ;RESTORE LNMB ADDRESS
                42 50   E9  0464  1944          BLBC    R0,40$                  ;EXIT ON ANY ERROR
                            0467  1945
          51    11 A1   9E  0467  1946 1$:      MOVAB   LNMB$T_NAME(R1),R1      ;ADDRESS OF NAME STRING
             50    81   9A  046B  1947          MOVZBL  (R1)+,R0                ;RETRIEVE SIZE OF NAME STRING
             51    50   C0  046E  1948          ADDL2   R0,R1                   ;POSITION TO FIRST LNMX
                            0471  1949
                   02   E0  0471  1950 5$:      BBS     #LNMX$V_XEND,-          ;IS THIS THE LAST TRANSLATION?
                12 61       0473  1951                  LNMX$B_FLAGS(R1),10$    ;IF SO, NO INDEX 0 LNMX SO RETURN ERROR
                01 A1   95  0475  1952          TSTB    LNMX$B_INDEX(R1)        ;IS THE INDEX 0 LNMX?
                   15   13  0478  1953          BEQL    20$                     ;IF SO, GO RETURN LNMX
                   0B   14  047A  1954          BGTR    10$                     ;IF POSITIVE INDEX THEN GO RETURN ERROR
          50    04 A1   9A  047C  1955          MOVZBL  LNMX$T_XLATION(R1),R0   ;ELSE RETRIEVE SIZE OF TRANSLATION
       51    05 A140   9E  0480  1956          MOVAB   LNMX$T_XLATION+1(R1)[R0],R1 ;POSITION TO NEXT LNMX
                   EA   11  0485  1957          BRB     5$                      ;GO SEE IF ITS INDEX IS 0
                            0487  1958
04 AE      01BC 8F   3C  0487  1959 10$:      MOVZWL  #SS$_NOLOGNAM,4(SP)     ;ELSE, RETURN THE APPROPRIATE ERROR
                   10   11  048D  1960          BRB     30$                     ;STATUS
                            048F  1961
          50    04 A1   9A  048F  1962 20$:      MOVZBL  LNMX$T_XLATION(R1),R0   ;RETRIEVE SIZE OF TRANSLATION STRING
             50    05   C0  0493  1963          ADDL2   #LNMX$T_XLATION+1,R0    ;ADD SIZE OF LNMX OVERHEAD + COUNT FIELD
                   54   DD  0496  1964          PUSHL   R4                      ;SAVE PCB ADDRESS
       66    61    50   28  0498  1965          MOVC3   R0,(R1),(R6)            ;MOVE ENTIRE LNMX FOR INDEX 0
                   54 8ED0  049C  1966          POPL    R4                      ;RESTORE PCB ADDRESS
                            049F  1967
                02DB   30  049F  1968 30$:      BSBW    LNM$UNLOCK              ;UNLOCK THE LOGICAL NAME MUTEX
                            04A2  1969          ENBINT                          ;RESTORE IPL TO ITS VALUE ON ENTRY
                   50 8ED0  04A5  1970          POPL    R0                      ;RESTORE STATUS
                      05    04A8  1971          RSB                             ;RETURN
04 AE             50   D0  04A9  1972 40$:      MOVL    R0,4(SP)                ;CHANGE RETURN STATUS TO AN ERROR STATUS
                  F0   11  04AD  1973          BRB     30$                     ;GO RETURN
                           04AF  1974 ;         .PAGE
```

```
                                                C 1
                                                                .SBTTL  LNM$FIRSTTAB    - SEARCH FOR FIRST TABLE NAME
                                                        ;+
                                                        ; LNM$FIRSTTAB - SEARCH FOR FIRST TABLE NAME
                                                        ;
                                                        ; THIS ROUTINE IS CALLED TO LOOKUP A LOGICAL NAME TABLE NAME.  THE FIRST MATCH
                                                        ; FOUND IS RETURNED.
                                                        ; ACCESS TO THE TABLE IS NOT CHECKED.
                                                        ;
                                                        ; INPUTS:
                                                        ;
                                                        ;       R1 = SEARCH ACCESS MODE IN LOW BYTE,
                                                        ;            CASELESS FLAG IN BIT 8,
                                                        ;            HIGH ORDER WORD 0.
                                                        ;       R2 = LENGTH OF TABLE NAME STRING.
                                                        ;       R3 = ADDRESS OF TABLE NAME STRING.
                                                        ;
                                                        ;       IT IS ASSUMED THAT THE LOGICAL NAME MUTEX IS LOCKED FOR AT LEAST READ
                                                        ;       ACCESS.
                                                        ;
                                                        ; OUTPUTS:
                                                        ;
                                                        ;       R0 LOW BIT CLEAR INDICATES SEARCH FAILURE.
                                                        ;
                                                        ;               R0 = SS$_NOLOGTAB - NO LOGICAL NAME TABLE NAME MATCH FOUND.
                                                        ;               R1 = JUNK.
                                                        ;
                                                        ;       R0 LOW BIT SET INDICATES SUCCESS WITH:
                                                        ;
                                                        ;               R1 = ADDRESS OF LOGICAL NAME TABLE HEADER.
                                                        ;
                                                        ;       REGISTERS R2 AND R3 ARE MODIFIED.
                                                        ;       REGISTERS R4 AND R5 ARE PRESERVED.
                                                        ;-
                                                        ;
                                                LNM$FIRSTTAB::                          ;SEARCH FOR LOGICAL NAME TABLE
                55    DD                        PUSHL   R5                              ;SAVE REGISTER
             5E 2C    C2                        SUBL    #RT_K_LENGTH-4,SP               ;ALLOCATE RECURSIVE TABLE NAME CONTROL BLOCK
                51    DD                        PUSHL   R1                              ;ACMODE, CASE FLAG
             55 5E    D0                        MOVL    SP,R5                           ;ADDRESS OF BLOCK
                0134  30                        BSBW    LNM$SETUP                       ;SETUP TABLE PROCESSING
             5E 30    C0                        ADDL2   #RT_K_LENGTH,SP                 ;REMOVE TABLE SEARCH CONTROL BLOCK
                55 8ED0                         POPL    R5                              ;RESTORE REGISTER
              05 50   E8                        BLBS    R0,10$                          ;RETURN IF SUCCESSFUL
          50  2294 8F  3C                      MOVZWL  #SS$_NOLOGTAB,R0                ;ELSE SETUP TO RETURN APPROPRIATE ERROR
                     05 04CA 2020 10$:          RSB                                     ;RETURN, STATUS IN R0
                        04CB 2021
                        04CB 2022 ;             .PAGE
```

```
                                D 1
04CB  2024          .SBTTL  LNM$PRESEARCH   - FIND FIRST CANDIDATE NAME
04CB  2025 ;+
04CB  2026 ; LNM$PRESEARCH - FIND FIRST CANDIDATE LOGICAL NAME
04CB  2027 ;
04CB  2028 ; THIS ROUTINE IS CALLED TO SEARCH A LOGICAL NAME HASH TABLE FOR THE FIRST
04CB  2029 ; CANDIDATE LOGICAL NAME MATCH.  IF A LOGICAL NAME TABLE IS SPECIFIED, THE
04CB  2030 ; SEARCH CONTINUES TO A SPECIFIC NAME.
04CB  2031 ;
04CB  2032 ; INPUTS:
04CB  2033 ;
04CB  2034 ;       R1 = HASH TABLE ADDRESS (IF TABLE HEADER ADDRESS IS MISSING)
04CB  2035 ;       R3 = ADDRESS OF NAME TRANSLATION (NT) BLOCK
04CB  2036 ;
04CB  2037 ;       NAME TABLE BLOCK REQUIREMENTS:
04CB  2038 ;
04CB  2039 ;               NT_W_RT         : MUST BE INITIALIZED
04CB  2040 ;               NT_W_HASH       : MAY BE INITIALIZED OR 0
04CB  2041 ;               NT_L_NAMLEN     : MUST BE INITIALIZED
04CB  2042 ;               NT_L_NAMADR     : MUST BE INITIALIZED
04CB  2043 ;               NT_L_TABID      : MAY BE INITIALIZED OR 0
04CB  2044 ;               NT_L_THREAD     : UNINITIALIZED
04CB  2045 ;
04CB  2046 ;       IT IS ASSUMED THAT THE LOGICAL NAME MUTEX IS LOCKED FOR AT LEAST READ
04CB  2047 ;       ACCESS.
04CB  2048 ;
04CB  2049 ; OUTPUTS:
04CB  2050 ;
04CB  2051 ;       R0 LOW BIT CLEAR INDICATES SEARCH FAILURE.
04CB  2052 ;
04CB  2053 ;               R0 = SS$_NOLOGNAM - NO LOGICAL NAME MATCH FOUND.
04CB  2054 ;               R1 = ADDRESS OF LOGICAL NAME BLOCK ON WHICH SEARCH FAILED.
04CB  2055 ;
04CB  2056 ;               NAME TABLE BLOCK REQUIREMENTS:
04CB  2057 ;
04CB  2058 ;                       NT_W_RT         : UNCHANGED
04CB  2059 ;                       NT_W_HASH       : VALID
04CB  2060 ;                       NT_L_NAMLEN     : UNCHANGED
04CB  2061 ;                       NT_L_NAMADR     : UNCHANGED
04CB  2062 ;                       NT_L_TABID      : UNCHANGED
04CB  2063 ;                       NT_L_THREAD     : ADDRESS OF LOGICAL NAME BLOCK
04CB  2064 ;                                         PRECEEDING THE BLOCK ON WHICH THE
04CB  2065 ;                                         SEARCH FAILED.
04CB  2066 ;
04CB  2067 ;       R0 LOW BIT SET INDICATES SUCCESS WITH:
04CB  2068 ;
04CB  2069 ;               R1 = ADDRESS OF LOGICAL NAME BLOCK THAT CONTAINS MATCH.
04CB  2070 ;
04CB  2071 ;               NAME TABLE BLOCK REQUIREMENTS:
04CB  2072 ;
04CB  2073 ;                       NT_W_RT         : UNCHANGED
04CB  2074 ;                       NT_W_HASH       : VALID
04CB  2075 ;                       NT_L_NAMLEN     : UNCHANGED
04CB  2076 ;                       NT_L_NAMADR     : UNCHANGED
04CB  2077 ;                       NT_L_TABID      : UNCHANGED
04CB  2078 ;                       NT_L_THREAD     : ADDRESS OF LOGICAL NAME BLOCK
04CB  2079 ;                                         PRECEEDING THE BLOCK ON WHICH THE
04CB  2080 ;                                         SEARCH SUCCEEDED.
```

```
                                04CB  2081 ;
                                04CB  2082 ;       R3,R4,R5 PRESERVED.
                                04CB  2083 ;-
                                04CB  2084         .ENABLE LSB
                                04CB  2085
                                04CB  2086 LNM$PRESEARCH::                              ;SEARCH FOR LOGICAL NAME
      50    0C A3    D0        04CB  2087         MOVL    NT_L_TABID(R3),R0            ;ADDRESS OF TABLE HEADER
            04       13        04CF  2088         BEQL    10$                          ;BRANCH IF NOT SPECIFIED
      51    01 A0    D0        04D1  2089         MOVL    LNMTH$L_HASH(R0),R1          ;HASH TABLE ADDRESS
      50    02 A3    3C        04D5  2090 10$:    MOVZWL  NT_W_HASH(R3),R0             ;HASH FUNCTION AVAILABLE?
            10       12        04D9  2091         BNEQ    20$                          ;BRANCH IF YES
            51       DD        04DB  2092         PUSHL   R1                           ;SAVE HASH TABLE ADDRESS
      50    04 A3    7D        04DD  2093         MOVQ    NT_L_NAMLEN(R3),R0           ;NAME DESCRIPTOR
             008A    30        04E1  2094         BSBW    LNM$HASH                     ;COMPUTE HASH FUNCTION
   02 A3    50       B0        04E4  2095         MOVW    R0,NT_W_HASH(R3)             ;SAVE HASH FUNCTION
            51    8ED0        04E8  2096         POPL    R1                           ;RESTORE HASH TABLE ADDRESS
      50    61       CA        04EB  2097 20$:    BICL2   LNMHSH$L_MASK(R1),R0         ;MASK OFF UNWANTED BITS OF HASH FUNCTION
51    0C A140       DE        04EE  2098         MOVAL   LNMHSH$K_BUCKET(R1)[R0],-    ;COMPUTE ADDRESS OF HASH BUCKET AND
                                04F3  2099                 R1                       ;STORE ITS ADDRESS AS ADDRESS OF
                                04F3  2100         ASSUME  LNMB$L_FLINK,EQ,0            ;PREVIOUS LNMB IN NT_L_THREAD
            0D       11        04F3  2101         BRB     30$                          ;RUN DOWN HASH CHAIN AND RETURN
                                04F5  2102
                                04F5  2103 ;       .PAGE
```

```
                        04F5  2105          .SBTTL  LNM$CONTSEARCH  - FIND NEXT CANDIDATE NAME
                        04F5  2106 ;+
                        04F5  2107 ; LNM$CONTSEARCH - FIND NEXT CANDIDATE LOGICAL NAME
                        04F5  2108 ;
                        04F5  2109 ; THIS ROUTINE IS CALLED TO SEARCH A LOGICAL NAME HASH BUCKET FOR THE NEXT
                        04F5  2110 ; CANDIDATE LOGICAL NAME MATCH.
                        04F5  2111 ;
                        04F5  2112 ; INPUTS:
                        04F5  2113 ;
                        04F5  2114 ;       R3 = ADDRESS NAME TABLE BLOCK.
                        04F5  2115 ;
                        04F5  2116 ;       NAME TABLE BLOCK REQUIREMENTS:
                        04F5  2117 ;
                        04F5  2118 ;               NT_W_RS         : MUST BE INITIALIZED
                        04F5  2119 ;               NT_W_HASH       : MUST BE INITIALIZED
                        04F5  2120 ;               NT_L_NAMLEN     : MUST BE INITIALIZED
                        04F5  2121 ;               NT_L_NAMADR     : MUST BE INITIALIZED
                        04F5  2122 ;               NT_L_TABID      : MUST BE INITIALIZED
                        04F5  2123 ;               NT_L_THREAD     : MUST BE INITIALIZED
                        04F5  2124 ;                       ADDRESS OF PREVIOUS LNMB$ BLOCK -
                        04F5  2125 ;                       SEARCH CONTINUES WITH THE FOLLOWING ENTRY.
                        04F5  2126 ;
                        04F5  2127 ;       IT IS ASSUMED THAT THE LOGICAL NAME MUTEX IS LOCKED FOR AT LEAST READ
                        04F5  2128 ;       ACCESS.
                        04F5  2129 ;
                        04F5  2130 ; OUTPUTS:
                        04F5  2131 ;
                        04F5  2132 ;       R0 LOW BIT CLEAR INDICATES SEARCH FAILURE.
                        04F5  2133 ;
                        04F5  2134 ;               R0 = SS$_NOLOGNAM - NO LOGICAL NAME MATCH FOUND.
                        04F5  2135 ;               R1 = ADDRESS OF LOGICAL NAME BLOCK ON WHICH SEARCH FAILED.
                        04F5  2136 ;
                        04F5  2137 ;               NT_L_THREAD CONTAINS ADDRESS OF PREVIOUS LNMB BLOCK
                        04F5  2138 ;
                        04F5  2139 ;       R0 LOW BIT SET INDICATES SUCCESS WITH:
                        04F5  2140 ;
                        04F5  2141 ;               R1 = ADDRESS OF LOGICAL NAME BLOCK THAT CONTAINS MATCH.
                        04F5  2142 ;
                        04F5  2143 ;               NT_L_THREAD CONTAINS ADDRESS OF PREVIOUS LNMB BLOCK
                        04F5  2144 ;
                        04F5  2145 ;       R3,R4,R5 ARE PRESERVED.
                        04F5  2146 ;-
                        04F5  2147
                        04F5  2148 LNM$CONTSEARCH::                             ;SEARCH FOR LOGICAL NAME
                        04F5  2149          ASSUME  LNMB$L_FLINK,EQ,0
    51    10 A3   DC    04F5  2150          MOVL    NT_L_THREAD(R3),R1          ;ADDRESS OF PREVIOUS LOGICAL NAME BLOCK
 09 63      0A   E0    04F9  2151          BBS     #NT_V_MODIFY,(R3),35$       ;CAN HASH BUCKET CHANGE?
    51      61   D0    04FD  2152          MOVL    LNMB$L_FLINK(R1),R1         ;IF NOT THEN RETRIEVE ADDRESS  NEXT LNMB
            54   11    0500  2153          BRB     55$                         ;AND SKIP FIRST CMPC3
                       0502  2154
    10 A3   51   D0    0502  2155 30$:     MOVL    R1,NT_L_THREAD(R3)          ;SAVE ADDRESS OF PREVIOUS BLOCK
    51      61   D0    0506  2156 35$:     MOVL    LNMB$L_FLINK(R1),R1         ;GET ADDRESS OF NEXT LOGICAL NAME BLOCK
            45   13    0509  2157          BEQL    70$                         ;BRANCH IF NO NEXT BLOCK
11 A1   04  A3   91    050B  2158          CMPB    NT_L_NAMLEN(R3), -          ;LENGTH'S MATCH?
                       0510  2159                  LNMB$T_NAME(R1)
            F0   1A    0510  2160          BGTRU   30$                         ;KEEP LOOKING DOWN CHAIN
            3C   1F    0512  2161          BLSSU   70$                         ;NO MATCH
```

```
                        53   DD  0514  2162          PUSHL   R3                      ;SAVE SEARCH PARAMETERS
                        51   DD  0516  2163          PUSHL   R1
        12 A1   08 B3   04 A3   29  0518  2164          CMPC3   NT_L_NAMLEN(R3), -      ;LOGICAL NAME STRINGS MATCH?
                                    051F  2165                  @NT_L_NAMADR(R3), -
                                    051F  2166                  LNMB$T_NAME+1(R1)
                22 04 BE    08  E1  051F  2167          BBC     #NT_V_CASE,@4(SP),50$   ;BRANCH IF CASED SEARCH
                        20   13  0524  2168 40$:       BEQL    50$                     ;BRANCH ON MATCH
                     50 81   9A  0526  2169          MOVZBL  (R1)+,R0                ;GET CASELESS CHARACTER
             7E 00000000'GF40   90  0529  2170          MOVB    G^EXE$UPCASE_DAT[R0],-(SP)
                     50 83   9A  0531  2171          MOVZBL  (R3)+,R0                ;GET CASELESS CHARACTER
          00000000'GF40 8E   91  0534  2172          CMPB    (SP)+,G^EXE$UPCASE_DAT[R0]
                        08   12  053C  2173          BNEQ    50$                     ;BRANCH IF DIFFERENT IN MORE THAN CASE
                        52   D7  053E  2174          DECL    R2                      ;PASS OVER THESE CHARACTERS
                 63 61  52   29  0540  2175          CMPC3   R2,(R1),(R3)            ;RESUME COMPARISON
                        DE   11  0544  2176          BRB     40$                     ;CONTINUE CASELESS COMPARISON
                        0A   BA  0546  2177 50$:       POPR    #^M<R1,R3>              ;RETRIEVE SEARCH PARAMETERS
                        0C   13  0548  2178          BEQLU   55$                     ;IF EQL FOUND IT
                        B6   1A  054A  2179          BGTRU   30$                     ;IF GTR KEEP SEARCHING
                   B2 63    08  E0  054C  2180          BBS     #NT_V_CASE,(R3),30$     ;BRANCH TO CONTINUE IF CASED SEARCH
                50 01BC 8F   3C  0550  2181 70$:       MOVZWL  #SS$_NOLOGNAM,R0        ;SET NO LOGICAL NAME MATCH
                        05   0555  2182          RSB                             ;EXIT
                             0556  2183
                  50 0C A3   D0  0556  2184 55$:       MOVL    NT_L_TABID(R3),R0       ;RETRIEVE TARGET TABLE HEADER ADDRESS
                        0E   13  055A  2185          BEQL    60$                     ;DONE IF IGNORING TABLE
                0C A1   50   D1  055C  2186          CMPL    R0,LNMB$L_TABLE(R1)     ;COMPARE TABLE
                        A0   1A  0560  2187          BGTRU   30$                     ;CONT IF NOT YET REACHED TARGET TABLE
                        EC   1F  0562  2188          BLSSU   70$                     ;DONE IF PASSED TARGET TABLE
                0B A1   63   91  0564  2189          CMPB    NT_B_ACMODE(R3), -      ;COMPARE ACCESS MODES
                                    0568  2190                  LNMB$B_ACMODE(R1)
                        98   1F  0568  2191          BLSSU   30$                     ;CONT IF NOT YET REACHED ACCESS MODE
                             056A  2192
                     50 01   D0  056A  2193 60$:       MOVL    #SS$_NORMAL,R0          ;MATCH FOUND
                        05   05  056D  2194          RSB                             ;EXIT
                             056E  2195
                             056E  2196          .DISABLE        LSB
                             056E  2197 ;        .PAGE
```

```
                                                     H 1
                     056E  2199          .SBTTL  LNM$HASH        - HASHING ALGORITHM
                     056E  2200 ;+
                     056E  2201 ; LNM$HASH - THE HASH FUNCTION
                     056E  2202 ;
                     056E  2203 ; INPUTS:
                     056E  2204 ;       R0 = LENGTH OF NAME.
                     056E  2205 ;       R1 = ADDRESS OF NAME.
                     056E  2206 ;
                     056E  2207 ; OUTPUTS:
                     056E  2208 ;       R0 = RETURN HASH FUNCTION.
                     056E  2209 ;       R1, R2 SCRATCHED.
                     056E  2210 ;       R3,R4,R5 PRESERVED.
                     056E  2211 ;-
                     056E  2212
                     056E  2213 LNM$HASH::
                 53  DD 056E  2214          PUSHL   R3                      ;SAVE REGISTERS
                 50  DD 0570  2215          PUSHL   R0
          53   50  FE 8F 78  0572  2216          ASHL    #-2,R0,R3               ;DIVIDE BY FOUR
                     12 13  0577  2217          BEQL    20$                     ;NOT EVEN THAT LONG
              20202020 8F CB  0579  2218 10$:     BICL3   #^A/AAAA/\^A/aaaa/,-    ;FETCH CASELESS CHARACTERS
                   52 81     057F  2219                  (R1)+,R2
                   50 52 CC  0581  2220          XORL    R2,R0                   ;XOR IN THE CHARACTERS
              50   50 09 9C  0584  2221          ROTL    #9,R0,R0                ;SCRAMBLE A BIT
                    EE 53 F5  0588  2222          SOBGTR  R3,10$                  ;LOOP UNTIL DONE
          53   8E  FFFFFFFC 8F CB  058B  2223 20$:     BICL3   #-4,(SP)+,R3            ;SAVE REMAINDER OF DIVIDE BY FOUR
                     0B 11  0593  2224          BRB     40$
              52   81   20 88  0595  2225 30$:     BICB3   #^A/A/\^A/a/,(R1)+,R2   ;FETCH A CASELESS CHARACTER
                   50   52 8C  0599  2226          XORB    R2,R0                   ;XOR IN A CHARACTER
              50   50   0D 9C  059C  2227          ROTL    #13,R0,R0               ;SCRAMBLE
                     F2 53 F4  05A0  2228 40$:     SOBGEQ  R3,30$                  ;LOOP UNTIL DONE
          50   71279461 8F C4  05A3  2229          MULL    #^X71279461,R0          ;MULT BY A FUNNY NUMBER
              50   50   13 9C  05AA  2230          ROTL    #32-13,R0,R0            ;GET SIGNIFICANT BITS
                 53 8ED0  05AE  2231          POPL    R3                      ;RESTORE REGISTERS
                     05 05B1  2232          RSB
                     05B2  2233 ;           .PAGE
```

```
                                                     I  1
                     05B2  2235          .SBTTL  LNM$LOOKUP      - LOOKUP TABLE NAME
                     05B2  2236 ;+
                     05B2  2237 ; LNM$LOOKUP - LOOKUP TABLE NAME
                     05B2  2238 ;
                     05B2  2239 ; THIS ROUTINE IS CALLED TO LOOKUP A LOGICAL NAME TABLE NAME.
                     05B2  2240 ;
                     05B2  2241 ; INPUTS:
                     05B2  2242 ;
                     05B2  2243 ;       R0 = HASH CODE VALUE OF LOGICAL NAME TABLE STRING (OR 0 IF NOT KNOWN)
                     05B2  2244 ;       R2 = LENGTH OF LOGICAL NAME TABLE STRING.
                     05B2  2245 ;       R3 = ADDRESS OF LOGICAL NAME TABLE STRING.
                     05B2  2246 ;       R5 = ADDRESS OF RECURSION TABLE SEARCH CONTROL BLOCK
                     05B2  2247 ;
                     05B2  2248 ;       IT IS ASSUMED THAT THE LOGICAL NAME MUTEX IS LOCKED FOR AT LEAST READ
                     05B2  2249 ;       ACCESS.
                     05B2  2250 ;
                     05B2  2251 ; OUTPUTS:
                     05B2  2252 ;
                     05B2  2253 ;       R1 POINTS TO THE NAME BLOCK.
                     05B2  2254 ;
                     05B2  2255 ; R2 AND R3 ARE MODIFIED.
                     05B2  2256 ;-
                     05B2  2257
                     05B2  2258 LNM$LOOKUP:
          55    DD   05B2  2259          PUSHL   R5                      ;SAVE REGISTER
          7E    7C   05B4  2260          CLRQ    -(SP)                   ;NO NAME BLOCK ADDRESS
                     05B6  2261                                          ;SPACE FOR DIRECTORY TABLE ADDRESS
       7E    52    7D  05B6  2262          MOVQ    R2,-(SP)                ;ADDRESS AND LENGTH OF NAME
       7E    50    B0  05B9  2263          MOVW    R0,-(SP)                ;HASH CODE VALUE OF TABLE NAME STRING
       7E    65    B0  05BC  2264          MOVW    RT_W_R5(R5),-(SP)       ;CASE FLAG AND ACCESS MODE
       53    5E    D0  05BF  2265          MOVL    SP,R3                   ;ADDRESS OF BLOCK
 55    00000000'EF    DE  05C2  2266          MOVAL   L^LNM$AL_DIRTBL,R5      ;ADDR OF SYSTEM DIRECTORY TABLE ADDRESS
       11 5E    1F    E0  05C9  2267          BBS     #31,SP,10$              ;BRANCH IF SMALL PROCESS
       50   04 B5    D0  05CD  2268          MOVL    @4(R5),R0               ;ADDRESS OF PROCESS DIRECTORY TABLE
             0B    13  05D1  2269          BEQL    10$                     ;SKIP IF NO TABLE DEFINED
          0C A0    D0  05D3  2270          MOVL    LNMB$L_TABLE(R0),-      ;ADDRESS OF TABLE HEADER OF PROCESS
          0C A3        05D6  2271                  NT_L_TABID(R3)          ;DIRECTORY TABLE
           FEF0   30  05D8  2272          BSBW    LNM$PRESEARCH           ;SEARCH PROCESS DIRECTORY
          0B 50    E8  05DB  2273          BLBS    R0,20$                  ;BRANCH IF NAME FOUND IN PROCESS SPACE
       50   95    D0  05DE  2274 10$:     MOVL    @(R5)+,R0               ;ADDRESS OF SYSTEM DIRECTORY TABLE
          0C A0    D0  05E1  2275          MOVL    LNMB$L_TABLE(R0),-      ;ADDRESS OF TABLE HEADER OF SYSTEM
          0C A3        05E4  2276                  NT_L_TABID(R3)          ;DIRECTORY TABLE
           FEE2   30  05E6  2277          BSBW    LNM$PRESEARCH           ;SEARCH SYSTEM DIRECTORY AND RETURN
       5E   14    C0  05E9  2278 20$:     ADDL    #NT_K_LENGTH,SP         ;FLUSH TRANSLATION BLOCK
             55 8ED0  05EC  2279          POPL    R5                      ;RESTORE REGISTER
                05    05EF  2280 LRSB:    RSB
                     05F0  2281
                     05F0  2282 ;         .PAGE
```

```
                        05F0  2284          .SBTTL  LNM$SETUP       - SETUP TO PROCESS TABLE NAME
                        05F0  2285 ;+
                        05F0  2286 ; LNM$SETUP - SETUP TO PROCESS LOGICAL NAME TABLE NAME
                        05F0  2287 ;
                        05F0  2288 ; THIS ROUTINE IS CALLED TO SETUP TO PROCESS A LOGICAL NAME TABLE NAME.
                        05F0  2289 ; TABLE SEARCHING IS INITIALIZED.
                        05F0  2290 ;
                        05F0  2291 ; INPUTS:
                        05F0  2292 ;
                        05F0  2293 ;       R2 = LENGTH OF LOGICAL NAME TABLE STRING.
                        05F0  2294 ;       R3 = ADDRESS OF LOGICAL NAME TABLE STRING.
                        05F0  2295 ;       R5 = ADDRESS OF RECURSIVE TABLE NAME TRANSLATION BLOCK
                        05F0  2296 ;            WITH RT_W_R5 FIELDS INITIALIZED
                        05F0  2297 ;
                        05F0  2298 ;       IT IS ASSUMED THAT THE LOGICAL NAME MUTEX IS LOCKED FOR AT LEAST READ
                        05F0  2299 ;       ACCESS.
                        05F0  2300 ;
                        05F0  2301 ; OUTPUTS:
                        05F0  2302 ;
                        05F0  2303 ;       R0 CONTAINS ERROR STATUS FROM SEARCHING.
                        05F0  2304 ;               R0 = SS$_NOLOGNAM - NO LOGICAL NAME MATCH FOUND.
                        05F0  2305 ;               R2,R3,R5 ARE MODIFIED.
                        05F0  2306 ;               R4 IS PRESERVED.
                        05F0  2307 ;
                        05F0  2308 ;       R0 LOW BIT SET INDICATES SUCCESS WITH:
                        05F0  2309 ;               R1 = ADDRESS OF LOGICAL NAME TABLE HEADER.
                        05F0  2310 ;               R2 = LENGTH OF ACTUAL LOGICAL NAME TABLE STRING
                        05F0  2311 ;               R3 = ADDRESS OF ACTUAL LOGICAL NAME TABLE STRING
                        05F0  2312 ;               R5 = ADDRESS OF CONTROL BLOCK.
                        05F0  2313 ;-
                        05F0  2314
                        05F0  2315 LNM$SETUP::
          02    8A      05F0  2316          BICB2   #RT_M_TERM,-            ;CLEAR LAST TRANSLATION BIT
       01 A5            05F2  2317                  RT_B_FLAGS(R5)
       02 A5    94      05F4  2318          CLRB    RT_B_DEPTH(R5)          ;INITIALIZE RECURSION DEPTH TO 0
       FF 8F    90      05F7  2319          MOVB    #RT_C_MAXTRIES,-        ;INITIALIZE MAXIMUM NUMBER OF TRIES
       03 A5            05FA  2320                  RT_B_TRIES(R5)
          50    D4      05FC  2321          CLRL    R0                      ;HASH CODE VALUE OF INITIAL TABLE NAME
                        05FE  2322                                          ;STRING IS NOT KNOWN
        FFB1    30      05FE  2323          BSBW    LNM$LOOKUP              ;LOOKUP THE INITIAL NAME
      EB 50     E9      0601  2324          BLBC    R0,LRSB                 ;NO SUCH NAME
08 A5    51     11      C1    0604  2325          ADDL3   #LNMB$T_NAME,R1,-     ;SAVE INITIAL LNMB IN RECURSION TABLE
                        0609  2326                  RT_L_STACK(R5)          ; AS THE STARTING POINT OF TRANSLATIONS
        00E4    30      0609  2327          BSBW    LNM$TBL_CACHE           ;CHECK THE TABLE TRANSLATION CACHE
   04 A5    50     D0    060C  2328          MOVL    R0,RT_L_CACHEPTR(R5)    ;SAVE CACHE POINTER
            57     13    0610  2329          BEQL    LNM$TABLE_SRCH          ;USE LONG WAY IF NO CACHE ENTRY
   0B A0    01     8E    0612  2330          MNEGB   #1,LNMC$B_CACHEINDX(R0) ;START WITH INITIAL ENTRY
                        0616  2331 ;        BRB     LNM$TABLE               ; AND DROP INTO LNM$TABLE
                        0616  2332
                        0616  2333 ;        .PAGE
```

```
                                                K  1
0616  2335          .SBTTL  LNM$TABLE       - PROCESS LOGICAL NAME TABLE
0616  2336 ;+
0616  2337 ; LNM$TABLE - PROCESS LOGICAL NAME TABLE NAME
0616  2338 ;
0616  2339 ; THIS ROUTINE IS CALLED TO PROCESS A LOGICAL NAME TABLE NAME.
0616  2340 ; THE TABLE NAME TRANSLATION CACHE IS USED IF POSSIBLE, ELSE
0616  2341 ; THE NAME IS RECURSIVELY TRANSLATED.  A CALLBACK IS PERFORMED
0616  2342 ; FOR EVERY TABLE THAT IS FOUND.
0616  2343 ;
0616  2344 ; A BASIC ASSUMPTION THAT THIS ROUTINE MAKES IS THAT IT IS CALLED FIRST
0616  2345 ; THROUGH LNM$SETUP TO INITIALIZE THE RECURSION TABLE BEFORE BEING CALLED
0616  2346 ; DIRECTLY SUBSEQUENT TIMES.
0616  2347 ;
0616  2348 ; INPUTS:
0616  2349 ;
0616  2350 ;       R5 = ADDRESS OF TABLE NAME TRANSLATION BLOCK
0616  2351 ;            ALL FIELDS OF THE BLOCK MUST BE INITIALIZED.
0616  2352 ;
0616  2353 ;       IT IS ASSUMED THAT THE LOGICAL NAME MUTEX IS LOCKED FOR AT LEAST READ
0616  2354 ;       ACCESS.
0616  2355 ;
0616  2356 ; OUTPUTS:
0616  2357 ;
0616  2358 ;       R0 CONTAINS ERROR STATUS FROM SEARCHING.
0616  2359 ;            R0 = SS$_NOLOGNAM - NO LOGICAL NAME MATCH FOUND.
0616  2360 ;            R2,R3,R5 ARE MODIFIED.
0616  2361 ;            R4 IS PRESERVED.
0616  2362 ;
0616  2363 ;       R0 LOW BIT SET INDICATES SUCCESS WITH:
0616  2364 ;            R1 = ADDRESS OF LOGICAL NAME TABLE HEADER.
0616  2365 ;            R2 = LENGTH OF ACTUAL LOGICAL NAME TABLE STRING
0616  2366 ;            R3 = ADDRESS OF ACTUAL LOGICAL NAME TABLE STRING
0616  2367 ;            R5 = ADDRESS OF CONTROL BLOCK.
0616  2368 ;
0616  2369 ;-
0616  2370
0616  2371 LNM$TABLE::
  52   04 A5   D0  0616  2372          MOVL    RT_L_CACHEPTR(R5),R2    ;GET CACHE POINTER
       4D 13   13  061A  2373          BEQL    LNM$TABLE_SRCH          ;USE RECURSIVE METHOD IF NO CACHE
     0B A2   96    061C  2374          INCB    LNMC$B_CACHEINDX(R2)    ;GO TO NEXT ENTRY
  53   0B A2   9A  061F  2375          MOVZBL  LNMC$B_CACHEINDX(R2),R3 ;EXTRACT INDEX NUMBER
     1A   53   91  0623  2376          CMPB    R3,#LNMC$K_NUM_ENTRIES  ;OFF THE END?
       3E   1E   13  0626  2377          BGEQU   40$                     ;NOPE, SO USE THIS ENTRY
  51   18 A243   D0  0628  2378          MOVL    LNMC$L_ENTRY(R2)[R3],R1 ;GET ENTRY
       10   13   13  062D  2379          BEQL    20$                     ;IS THERE ONE?
                    062F  2380
  50   51   01   C1  062F  2381          ADDL3   #1,R1,R0                ;WAS THIS THE END FLAG?
       04   13   13  0633  2382          BEQL    10$                     ;THEN WE GOT ONE
     50   01   3C  0635  2383          MOVZWL  #SS$_NORMAL,R0          ;SUCCESS!
            05   05  0638  2384          RSB
                    0639  2385
  50   01BC 8F   3C  0639  2386 10$:     MOVZWL  #SS$_NOLOGNAM,R0        ;RETURN TABLES ALL DONE
            05   05  063E  2387          RSB
                    063F  2388
                    063F  2389 ;
                    063F  2390 ; NO ENTRY - TWO POSSIBLE CASES:
                    063F  2391 ;  1.   WE JUST RAN OFF THE END OF THE VALID ONES AND
```

```
                                                L 1
                     063F  2392 ;       NEED TO GO BACK TO THE BEGINNING AND REBUILD IT ALL.
                     063F  2393 ;       IN THIS CASE THE RECURSION TABLE IS STILL IN THE INITIALIZED STATE
                     063F  2394 ;       AND THE RECURSION DEPTH IS ZERO.
                     063F  2395 ;
                     063F  2396 ;  2.   WE ARE BUILDING THEM AS WE GO.
                     063F  2397 ;       IN THIS CASE THE RECURSION TABLE IS CURRENT AND THE CACHE INDEX
                     063F  2398 ;       IS CORRECT.
                     063F  2399 ;
                     063F  2400
          02 A5  95  063F  2401 20$:    TSTB    RT_B_DEPTH(R5)              ;RECURSION DEPTH 0?
             0E  12  0642  2402         BNEQ    30$                         ;NOPE
          0C A2  C1  0644  2403         ADDL3   LNMC$L_TBLADDR(R2),-        ;GET POINTER TO TABLE NAME
        51     11        0647  2404                 #LNMB$T_NAME,R1
    08 A5     51  D1  0649  2405         CMPL    R1,RT_L_STACK(R5)           ;INITIAL STATE?
             03  12  064D  2406         BNEQ    30$                         ;NO, SO PROCEED
          0B A2  94  064F  2407         CLRB    LNMC$B_CACHEINDX(R2)        ;GO BACK TO START
             15  10  0652  2408 30$:    BSBB    LNM$TABLE_SRCH              ;FIND NEXT (OR FIRST)
  52     04 A5  D0  0654  2409         MOVL    RT_L_CACHEPTR(R5),R2        ;GET CACHE POINTER
  53     0B A2  9A  0658  2410         MOVZBL  LNMC$B_CACHEINDX(R2),R3     ;EXTRACT INDEX NUMBER
18 A243     51  D0  065C  2411         MOVL    R1,LNMC$L_ENTRY(R2)[R3]     ;STORE TABLE HEADER ADDR
      1C A243  D4  0661  2412         CLRL    LNMC$L_ENTRY+4(R2)[R3]      ;CLEAR NEXT
             05  0665  2413         RSB                                 ;RETURN
                 0666  2414
          04 A5  D4  0666  2415 40$:    CLRL    RT_L_CACHEPTR(R5)           ;GIVE UP ON THE CACHE
                 0669  2416 ;       BRB     LNM$TABLE_SRCH              ;AND USE THE LONG METHOD
                 0669  2417
                 0669  2418 ;       .PAGE
```

```
                     0669  2420          .SBTTL  LNM$TABLE_SRCH  - PROCESS LOGICAL NAME TABLE
                     0669  2421 ;+
                     0669  2422 ; LNM$TABLE_SRCH - PROCESS LOGICAL NAME TABLE NAME
                     0669  2423 ;
                     0669  2424 ; THIS ROUTINE IS CALLED TO PROCESS A LOGICAL NAME TABLE NAME.
                     0669  2425 ; THE NAME IS RECURSIVELY TRANSLATED.  A CALLBACK IS PERFORMED
                     0669  2426 ; FOR EVERY TABLE THAT IS FOUND.
                     0669  2427 ;
                     0669  2428 ; A BASIC ASSUMPTION THAT THIS ROUTINE MAKES IS THAT IT IS CALLED FIRST
                     0669  2429 ; THROUGH LNM$SETUP TO INITIALIZE THE RECURSION TABLE BEFORE BEING CALLED
                     0669  2430 ; DIRECTLY SUBSEQUENT TIMES.
                     0669  2431 ;
                     0669  2432 ; INPUTS:
                     0669  2433 ;
                     0669  2434 ;       R5 = ADDRESS OF TABLE NAME TRANSLATION BLOCK
                     0669  2435 ;            ALL FIELDS OF THE BLOCK MUST BE INITIALIZED.
                     0669  2436 ;
                     0669  2437 ;       IT IS ASSUMED THAT THE LOGICAL NAME MUTEX IS LOCKED FOR AT LEAST READ
                     0669  2438 ;       ACCESS.
                     0669  2439 ;
                     0669  2440 ; OUTPUTS:
                     0669  2441 ;
                     0669  2442 ;       R0 CONTAINS ERROR STATUS FROM SEARCHING.
                     0669  2443 ;            R0 = SS$_NOLOGNAM - NO LOGICAL NAME MATCH FOUND.
                     0669  2444 ;            R2,R3,R5 ARE MODIFIED.
                     0669  2445 ;            R4 IS PRESERVED.
                     0669  2446 ;
                     0669  2447 ;       R0 LOW BIT SET INDICATES SUCCESS WITH:
                     0669  2448 ;            R1 = ADDRESS OF LOGICAL NAME TABLE HEADER.
                     0669  2449 ;            R2 = LENGTH OF ACTUAL LOGICAL NAME TABLE STRING
                     0669  2450 ;            R3 = ADDRESS OF ACTUAL LOGICAL NAME TABLE STRING
                     0669  2451 ;            R5 = ADDRESS OF CONTROL BLOCK.
                     0669  2452 ;
                     0669  2453 ;-
                     0669  2454
                     0669  2455 LNM$TABLE_SRCH:
  01 A5    02    8A  0669  2456 10$:    BICB2   #RT_M_TERM,RT_B_FLAGS(R5) ;CLEAR TERMINAL SEEN BIT
  50   02 A5    98  066D  2457         CVTBL   RT_B_DEPTH(R5),R0         ;RECURSION DEPTH
       65    19     0671  2458         BLSS    60$                       ;BRANCH IF NOTHING LEFT TO SCAN
51   08 A540   D0  0673  2459         MOVL    RT_L_STACK(R5)[R0],R1     ;ADDRESS OF PREVIOUS TRANSLATION STRING
                     0678  2460
     03 A5     97  0678  2461 20$:    DECB    RT_B_TRIES(R5)            ;DECREMENT NUMBER OF TRIES LEFT
        64     13  067B  2462         BEQL    70$                       ;DONE IF TRIES REACHES ZERO
                     067D  2463
     50    81    9A  067D  2464         MOVZBL  (R1)+,R0                  ;LENGTH OF PREVIOUS TRANSLATION STRING
     51    50    C0  0680  2465         ADDL2   R0,R1                     ;POSITION PAST PREVIOUS TRANSLATION
                     0683  2466
        02    E0  0683  2467         BBS     #LNMX$V_XEND,-            ;LAST TRANSLATION?
     4C 61              0685  2468                 LNMX$B_FLAGS(R1),50$      ;YES - GO DECREMENT DEPTH AND CONTINUE
52   01 A1    98  0687  2469         CVTBL   LNMX$B_INDEX(R1),R2       ;GET TRANSLATION INDEX
        34    19  068B  2470         BLSS    40$                       ;BRANCH IF SPECIAL VALUE
57 65    09    E0  068D  2471         BBS     #RT_V_TERM,(R5),80$       ;ERROR IF EXPECTING TABLE HEADER
                     0691  2472
        0A    91  0691  2473         CMPB    #LNM$C_MAXDEPTH,-         ;RECURSED TOO DEEPLY?
     02 A5              0693  2474                 RT_B_DEPTH(R5)            ;ERROR OF OVERLY DEEP
        4A    15  0695  2475         BLEQ    70$
                     0697  2476
```

```
                                                N  1

                50   02 A5   98   0697  2477          CVTBL   RT_B_DEPTH(R5),R0       ;RECURSION DEPTH
                53   04 A1   9E   069B  2478          MOVAB   LNMX$T_XLATION(R1),R3   ;TRANSLATION COUNTED STRING ADDRESS
           08 A540    53   D0   069F  2479          MOVL    R3,RT_C_STACK(R5)[R0]   ;SAVE RECURSION INFO BEFORE LOOKUP
                 02 A5   96   06A4  2480          INCB    RT_B_DEPTH(R5)          ;INCREMENT RECURSION DEPTH FOR NEXT TIME
              52    83   9A   06A7  2481          MOVZBL  (R3)+,R2                ;TRANSLATION STRING LENGTH AND ADDRESS
           50   02 A1   3C   06AA  2482          MOVZWL  LNMX$W_HASH(R1),R0      ;RETRIEVE TRANSLATION'S HASH CODE VALUE
                                   06AE  2483
                    01   E1   06AE  2484          BBC     #LNMX$V_TERMINAL,-      ;IS THIS TRANSLATION MARKED TERMINAL?
                 04 61           06B0  2485                  LNMX$B_FLAGS(R1),30$    ;NO  - GO DO LOOKUP
                    02   88   06B2  2486          BISB2   #RT_M_TERM,-            ;YES - ALLOW ONE AND ONLY ONE MORE
                 01 A5           06B4  2487                  RT_B_FLAGS(R5)          ;       TRANSLATION
                                   06B6  2488
                  FEF9   30   06B6  2489 30$:     BSBW    LNM$LOOKUP              ;LOOKUP LOGICAL NAME TABLE NAME
                17 50    E9   06B9  2490          BLBC    R0,50$                  ;GO DECR DEPTH AND CONTINUE IF NOT FOUND
              51    11    C0   06BC  2491          ADDL2   #LNMB$T_NAME,R1         ;POINT TO COUNTED NAME STRING
                    B7   11   06BF  2492          BRB     20$                     ;CONTINUE WITH FIRST TRANSLATION
                                   06C1  2493
              51    04    C0   06C1  2494 40$:     ADDL2   #LNMX$T_XLATION,R1      ;ADDRESS OF COUNTED TRANSLATION STRING
           82 8F    52   91   06C4  2495          CMPB    R2,#LNMX$C_TABLE        ;TABLE NAME?
                    AE   12   06C8  2496          BNEQ    20$                     ;IGNORE THIS TRANSLATION
                    51   D6   06CA  2497          INCL    R1                      ;POSITION TO TABLE HEADER
                 02 A5   97   06CC  2498          DECB    RT_B_DEPTH(R5)          ;DECREMENT RECURSION DEPTH
              50    01    D0   06CF  2499          MOVL    #SS$_NORMAL,R0          ;NORMAL STATUS
                          05   06D2  2500          RSB                             ;RETURN WITH R1 = TABLE HEADER ADDRESS
                                   06D3  2501
                 02 A5   97   06D3  2502 50$:     DECB    RT_B_DEPTH(R5)          ;DECREMENT RECURSION DEPTH
                    91   18   06D6  2503          BGEQ    10$                     ;RESUME
                                   06D8  2504
              51    01    CE   06D8  2505 60$:     MNEGL   #1,R1                   ;FLAG LAST TABLE
           50   01BC 8F    3C   06DB  2506          MOVZWL  #SS$_NOLOGNAM,R0        ;HAVE SCANNED ALL TABLE
                          05   06E0  2507          RSB
           50   0374 8F    3C   06E1  2508 70$:     MOVZWL  #SS$_TOOMANYLNAM,R0     ;RECURSION TOO DEEP
                    05   11   06E6  2509          BRB     90$                     ;RETURN
           50   015C 8F    3C   06E8  2510 80$:     MOVZWL  #SS$_IVLOGTAB,R0        ;INVALID TABLE NAME
                    51   D4   06ED  2511 90$:     CLRL    R1
                          05   06EF  2512          RSB
                                   06F0  2513
                                   06F0  2514 ;       .PAGE
```

```
                                                B 2
                     06F0  2516          .SBTTL  LNM$TBL_CACHE   - SEARCH LOGICAL NAME TABLE TRANSLATION CACHE
                     06F0  2517 ;+
                     06F0  2518 ; LNM$TBL_CACHE - SEARCH LOGICAL NAME TABLE TRANSLATION CACHE
                     06F0  2519 ;
                     06F0  2520 ; THIS ROUTINE IS CALLED TO SEARCH THE LOGICAL NAME TABLE NAME
                     06F0  2521 ; TRANSLATION CACHE.  IF A CACHE ENTRY EXISTS FOR THIS LOGICAL NAME,
                     06F0  2522 ; THEN THE ENTRY IS VALIDATED AND RETURNED.   IF INVALID OR NO ENTRY,
                     06F0  2523 ; A NEW ENTRY IS SELECTED, INITIALIZED AND RETURNED.
                     06F0  2524 ;
                     06F0  2525 ; INPUTS:
                     06F0  2526 ;
                     06F0  2527 ;       R1 = ADDRESS OF TABLE NAME LOGICAL NAME BLOCK
                     06F0  2528 ;
                     06F0  2529 ;       IT IS ASSUMED THAT THE LOGICAL NAME MUTEX IS LOCKED FOR AT LEAST READ
                     06F0  2530 ;       ACCESS.
                     06F0  2531 ;
                     06F0  2532 ; OUTPUTS:
                     06F0  2533 ;
                     06F0  2534 ;       R0 CONTAINS ADDRESS OF CACHE ENTRY TO USE OR 0
                     06F0  2535 ;          CACHE ENTRY MAY OR MAY NOT BE VALID
                     06F0  2536 ;
                     06F0  2537 ;-
                     06F0  2538
                     06F0  2539 LNM$TBL_CACHE:
          50 5E  1F  E0  06F0  2540          BBS     #31,SP,100$             ;NO CACHE IF NO P1 SPACE
                     06F4  2541
     52   00000000'9F  9E  06F4  2542          MOVAB   @#CTL$GQ_LNMTBLCACHE,R2 ;GET QUEUE HEADER ADDR
             50  52  D0  06FB  2543          MOVL    R2,R0                   ;POINT TO FIRST ENTRY
             50  60  D0  06FE  2544 10$:     MOVL    LNMC$L_FLINK(R0),R0     ;GET NEXT ENTRY
             50  52  D1  0701  2545          CMPL    R2,R0                   ;BACK TO QUEUE HEADER?
                 38  13  0704  2546          BEQL    80$                     ;YES, THEN MAKE A NEW ONE
          0C A0  51  D1  0706  2547          CMPL    R1,LNMC$L_TBLADDR(R0)   ;IS THIS THE ONE?
                 F2  12  070A  2548          BNEQ    10$                     ;TRY NEXT
                     070C  2549
             50  60  0F  070C  2550          REMQUE  (R0),R0                 ;REMOVE ENTRY
              10 A0  D1  070F  2551          CMPL    LNMC$L_PROCDIRSEQ(R0),- ;IS PROCESS DIRECTORY VALID?
     00000000'9F             0712  2552                  @#CTL$GL_LNMDIRSEQ
                 0A  12  0717  2553          BNEQ    40$                     ;NOPE, MUST RE-INIT ENTRY
              14 A0  D1  0719  2554          CMPL    LNMC$L_SYSDIRSEQ(R0),-  ;IS SYSTEM DIRECTORY VALID?
     00000000'9F             071C  2555                  @#LNM$GL_SYSDIRSEQ
                 17  13  0721  2556          BEQL    50$                     ;A CACHE HIT!
       0C A0     51  D0  0723  2557 40$:     MOVL    R1,LNMC$L_TBLADDR(R0)   ;SET TABLE NAME ADDR
     00000000'9F     D0  0727  2558          MOVL    @#CTL$GL_LNMDIRSEQ,-    ;SET NEW PROCESS DIRECTORY SEQ NUM.
              10 A0          072D  2559                  LNMC$L_PROCDIRSEQ(R0)
     00000000'9F     D0  072F  2560          MOVL    @#LNM$GL_SYSDIRSEQ,-    ;SET NEW SYSTEM DIRECTORY SEQ NUM.
              14 A0          0735  2561                  LNMC$L_SYSDIRSEQ(R0)
              18 A0  7C  0737  2562          CLRQ    LNMC$L_ENTRY(R0)        ;CLEAR FIRST ENTRIES
          62  60  0E  073A  2563 50$:     INSQUE  (R0),(R2)               ;INSERT AT HEAD OF QUEUE
                 05  073D  2564          RSB
                     073E  2565
       50  04 B2  0F  073E  2566 80$:     REMQUE  @LNMC$L_BLINK(R2),R0    ;TAKE OLDEST ENTRY
                 DF  1C  0742  2567          BVC     40$                     ;IF THERE ARE ANY
                     50  D4  0744  2568 100$:    CLRL    R0                      ;NO ENTRY
                 05  0746  2569          RSB
                     0747  2570
                     0747  2571 ;         .PAGE
```

```
                     0747  2573          .SBTTL  LNM$PROBER      - PROBE LOGICAL NAME DESCRIPTOR FOR READ ACCESS
                     0747  2574 ;+
                     0747  2575 ; LNM$PROBER - PROBE LOGICAL NAME DESCRIPTOR FOR READ ACCESS
                     0747  2576 ;
                     0747  2577 ; THIS ROUTINE IS CALLED TO PROBE A DESCRIPTOR FOR A LOGICAL NAME FOR READ
                     0747  2578 ; ACCESS.  IF CHECK THE LENGTH OF THE DESCRIPTOR FOR VALIDITY AS WELL
                     0747  2579 ; AS CHECKING ACCESS TO THE DESCRIBED BUFFER.
                     0747  2580 ; ACCESS TO THE DESCRIPTOR IS NOT CHECKED.
                     0747  2581 ;
                     0747  2582 ; INPUTS:
                     0747  2583 ;
                     0747  2584 ;       R0 = ADDRESS OF LOGICAL NAME STRING DESCRIPTOR
                     0747  2585 ;
                     0747  2586 ; OUTPUTS:
                     0747  2587 ;
                     0747  2588 ;       R0 LOW BIT CLEAR INDICATES FAILURE TO TRANSLATE.
                     0747  2589 ;
                     0747  2590 ;               R0 = SS$_ACCVIO - ACCESS VIOLATION.
                     0747  2591 ;               R0 = SS$_IVLOGNAM - INVALID LOGICAL NAME.
                     0747  2592 ;                    R1 AND R2 ARE MODIFIED.
                     0747  2593 ;
                     0747  2594 ;       R0 LOW BIT SET INDICATES SUCCESS.
                     0747  2595 ;
                     0747  2596 ;               R0 = SS$_NORMAL - DESCRIPTOR IS VALID
                     0747  2597 ;                    R1 = LENGTH OF BUFFER IN BYTES.
                     0747  2598 ;                    R2 = ADDRESS OF BUFFER.
                     0747  2599 ;-
                     0747  2600 LNM$PROBER::
     51    60    7D  0747  2601          MOVQ    (R0),R1                 ;FETCH DESCRIPTOR
     51    51    3C  074A  2602          MOVZWL  R1,R1                   ;GET LENGTH OF LOGICAL NAME STRING
           11    13  074D  2603          BEQL    10$                     ;IF EQL INVALID LOGICAL NAME
                     074F  2604          ASSUME  LNM$C_NAMLENGTH LE 512
00FF 8F    51    B1  074F  2605          CMPW    R1,#LNM$C_NAMLENGTH     ;LEGAL NAME STRING LENGTH?
           0A    1A  0754  2606          BGTRU   10$                     ;IF LEQU YES
                     0756  2607          IFNORD  R1,(R2),20$             ;CAN LOGICAL NAME STRING BE READ?
     50    01    D0  075C  2608          MOVL    #SS$_NORMAL,R0          ;SUCCESS STATUS
           05    05  075F  2609          RSB
50   0154 8F    3C  0760  2610 10$:     MOVZWL  #SS$_IVLOGNAM,R0        ;SET INVALID LOGICAL NAME
           05    05  0765  2611          RSB
     50    0C    3C  0766  2612 20$:     MOVZWL  #SS$_ACCVIO,R0          ;SET ACCESS VIOLATION
           05    05  0769  2613          RSB
                     076A  2614
                     076A  2615 ;       .PAGE
```

```
                                                D 2
                     076A  2617          .SBTTL  LNM$LOCKR       - LOCK LOGICAL NAME TABLE FOR READ ACCESS
                     076A  2618          .SBTTL  LNM$LOCKW       - LOCK LOGICAL NAME TABLE FOR WRITE ACCESS
                     076A  2619 ;+
                     076A  2620 ; LNM$LOCKR - LOCK LOGICAL NAME TABLE FOR READ ACCESS
                     076A  2621 ; LNM$LOCKW - LOCK LOGICAL NAME TABLE FOR WRITE ACCESS
                     076A  2622 ;
                     076A  2623 ; THESE ROUTINES ARE CALLED TO SYNCHRONIZE ACCESS TO LOGICAL NAME TABLES.
                     076A  2624 ;
                     076A  2625 ; INPUTS:
                     076A  2626 ;
                     076A  2627 ;       R4 = CURRENT PROCESS PCB ADDRESS.
                     076A  2628 ;
                     076A  2629 ; OUTPUTS:
                     076A  2630 ;
                     076A  2631 ;       REGISTER R0 IS MODIFIED.
                     076A  2632 ;       REGISTERS R1, R2, AND R3 ARE PRESERVED ACROSS CALL.
                     076A  2633 ;-
                     076A  2634
                     076A  2635          .ENABL  LSB
                     076A  2636 LNM$LOCKR::                              ;LOCK LOGICAL NAME TABLE FOR READ ACCESS
00000000'EF    9F    076A  2637          PUSHAB  L^SCH$LOCKR             ;SET ADDRESS OF LOCK ROUTINE
          0E   11    0770  2638          BRB     10$                     ;
                     0772  2639 LNM$LOCKW::                              ;LOCK LOGICAL NAME TABLE FOR WRITE ACCESS
00000000'EF    9F    0772  2640          PUSHAB  L^SCH$LOCKW             ;SET ADDRESS OF LOCK ROUTINE
          06   11    0778  2641          BRB     10$                     ;
                     077A  2642 ;        .PAGE
```

```
                     077A  2644          .SBTTL  LNM$UNLOCK      - UNLOCK LOGICAL NAME TABLE
                     077A  2645 ;+
                     077A  2646 ; LNM$UNLOCK - UNLOCK NAME TABLE
                     077A  2647 ;
                     077A  2648 ; THIS ROUTINE IS CALLED TO UNLOCK LOGICAL NAME TABLES AND ALLOW ACCESS BY
                     077A  2649 ; OTHER PROCESSES.
                     077A  2650 ;
                     077A  2651 ; INPUTS:
                     077A  2652 ;
                     077A  2653 ;       R4 = CURRENT PROCESS PCB ADDRESS.
                     077A  2654 ;
                     077A  2655 ; OUTPUTS:
                     077A  2656 ;
                     077A  2657 ;       R0, R1, R2, AND R3 ARE MODIFIED.
                     077A  2658 ;
                     077A  2659 ;-
                     077A  2660
                     077A  2661 LNM$UNLOCK::                                ;UNLOCK NAME TABLE
    00000000'EF   9F 077A  2662          PUSHAB  L^SCH$UNLOCK               ;SET ADDRESS OF LOCK ROUTINE
50  00000000'EF   DE 0780  2663 10$:     MOVAL   L^LNM$AL_MUTEX,R0          ;GET ADDRESS OF LOGICAL NAME TABLE MUTEX
                  05 0787  2664          RSB                                ;PERFORM SYNCHRONIZATION OPERATION
                     0788  2665          .DSABL  LSB
                     0788  2666
                     0788  2667          .END
```

```
ARM$M_READ                      = 00000001
ARM$M_WRITE                     = 00000002
CA$_MEASURE                     = 00000002
CHPCTL$B_MODE                   = 00000008
CHPCTL$C_LENGTH                 = 0000000C
CHPCTL$L_ACCESS                 = 00000000
CHPCTL$L_FLAGS                  = 00000004
CHPCTL$M_READ                   = 00000001
CHPCTL$M_USEREADALL             = 00000004
CHPCTL$M_WRITE                  = 00000002
CTL$GL_LNMDIRSEQ                ********    X   02
CTL$GQ_LNMTBLCACHE              ********    X   02
DELETE_ENTRY                    00000000 R      02
DELETE_LNMB                     0000004F R      02
DELETE_NAMES                    000000C4 R      02
DELETE_TABLE                    00000101 R      02
DYN$C_ORB                       = 00000049
EXE$CAKPRO_INT                  ********    X   02
EXE$CLEANUP_ORB                 ********    X   02
EXE$DEAP1                       ********    X   02
EXE$DEAPAGED                    ********    X   02
EXE$UPCASE_DAT                  ********    X   02
LNM$AL_DIRTBL                   ********    X   02
LNM$AL_HASHTBL                  ********    X   02
LNM$AL_MUTEX                    ********    X   02
LNM$CHECK_PROT                  00000131 RG     02
LNM$CONTSEARCH                  000004F5 RG     02
LNM$C_MAXDEPTH                  = 0000000A
LNM$C_NAMLENGTH                 = 000000FF
LNM$DELBLK                      00000192 RG     02
LNM$DELETE                      000001A6 R      02
LNM$DELETE_HASH                 000001FF RG     02
LNM$DELETE_LNMB                 000001BF RG     02
LNM$DELETE_TAB                  0000021A RG     02
LNM$FIRSTTAB                    000004AF RG     02
LNM$GL_SYSDIRSEQ                ********      X 02
LNM$HASH                        0000056E RG     02
LNM$INIT_PROT                   0000022B RG     02
LNM$INSLOGTAB                   0000026F RG     02
LNM$LOCKR                       0000076A RG     02
LNM$LOCKW                       00000772 RG     02
LNM$LOOKUP                      000005B2 R      02
LNM$PRESEARCH                   000004CB RG     02
LNM$PROBER                      00000747 RG     02
LNM$SEARCHLOG                   000003A5 RG     02
LNM$SEARCH_ONE                  00000438 RG     02
LNM$SETUP                       000005F0 RG     02
LNM$TABLE                       00000616 RG     02
LNM$TABLE_SRCH                  00000669 R      02
LNM$TBL_CACHE                   000006F0 R      02
LNM$UNLOCK                      0000077A RG     02
LNM$V_CREATE_IF                 = 00000018
LNMB$B_ACMODE                   = 0000000B
LNMB$B_FLAGS                    = 00000010
LNMB$L_BLINK                    = 00000004
LNMB$L_FLINK                    = 00000000
LNMB$L_TABLE                    = 0000000C
LNMB$T_NAME                     = 00000011
LNMB$V_NODELETE                 = 00000004
LNMB$V_NO_ALIAS                 = 00000000
LNMB$V_TABLE                    = 00000003
LNMB$W_SIZE                     = 00000008
LNMC$B_CACHEINDX                = 0000000B
LNMC$K_NUM_ENTRIES              = 0000001A
LNMC$L_BLINK                    = 00000004
LNMC$L_ENTRY                    = 00000018
LNMC$L_FLINK                    = 00000000
LNMC$L_PROCDIRSEQ               = 00000010
LNMC$L_SYSDIRSEQ                = 00000014
LNMC$L_TBLADDR                  = 0000000C
LNMHSH$K_BUCKET                 = 0000000C
LNMHSH$L_MASK                   = 00000000
LNMTH$B_FLAGS                   = 00000000
LNMTH$L_BYTES                   = 00000021
LNMTH$L_BYTESLM                 = 0000001D
LNMTH$L_CHILD                   = 00000011
LNMTH$L_HASH                    = 00000001
LNMTH$L_NAME                    = 00000009
LNMTH$L_ORB                     = 00000005
LNMTH$L_PARENT                  = 0000000D
LNMTH$L_QTABLE                  = 00000019
LNMTH$L_SIBLING                 = 00000015
LNMTH$V_DIRECTORY               = 00000001
LNMTH$V_GROUP                   = 00000002
LNMTH$V_SYSTEM                  = 00000003
LNMX$B_FLAGS                    = 00000000
LNMX$B_INDEX                    = 00000001
LNMX$C_BACKPTR                  = FFFFFF81
LNMX$C_TABLE                    = FFFFFF82
LNMX$T_XLATION                  = 00000004
LNMX$V_TERMINAL                 = 00000001
LNMX$V_XEND                     = 00000002
LNMX$W_HASH                     = 00000002
LRSB                            000005EF R      02
NT_B_ACMODE                     = 00000000
NT_B_FLAGS                      = 00000001
NT_K_LENGTH                     = 00000014
NT_L_NAMADR                     = 00000008
NT_L_NAMLEN                     = 00000004
NT_L_TABID                      = 0000000C
NT_L_THREAD                     = 00000010
NT_M_CASE                       = 00000001
NT_M_MODIFY                     = 00000004
NT_V_CASE                       = 00000008
NT_V_MODIFY                     = 0000000A
NT_W_HASH                       = 00000002
NT_W_RS                         = 00000000
ORB$B_FLAGS                     = 0000000B
ORB$B_TYPE                      = 0000000A
ORB$C_LENGTH                    = 00000058
ORB$K_LENGTH                    = 00000058
ORB$L_ACL_COUNT                 = 00000028
ORB$L_ACL_DESC                  = 0000002C
ORB$L_ACL_MUTEX                 = 00000004
```

```
ORB$L_GRP_PROT                  = 00000020
ORB$L_OWNER                     = 00000000
ORB$L_OWN_PROT                  = 0000001C
ORB$L_SYS_PROT                  = 00000018
ORB$L_WOR_PROT                  = 00000024
ORB$Q_MODE_PROT                 = 00000010
ORB$R_MAX_CLASS                 = 00000044
ORB$R_MIN_CLASS                 = 00000030
ORB$S_MAX_CLASS                 = 00000014
ORB$S_MIN_CLASS                 = 00000014
ORB$W_REFCOUNT                  = 0000000E
ORB$W_SIZE                      = 00000008
PCB$L_ARB                       = 0000008C
PCB$L_UIC                       = 000000BC
PCB$Q_PRIV                      = 00000084
PMS$GL_LOGNAM                     ********    X   02
PRS_IPL                           ********    X   02
PRV$V_GRPNAM                    = 00000003
PRV$V_SYSNAM                    = 00000002
PSL$C_USER                      = 00000003
RT_B_ACMODE                     = 00000000
RT_B_DEPTH                      = 00000002
RT_B_FLAGS                      = 00000001
RT_B_TRIES                      = 00000003
RT_C_MAXTRIES                   = 000000FF
RT_K_LENGTH                     = 00000030
RT_L_CACHEPTR                   = 00000004
RT_L_STACK                      = 00000008
RT_M_CASE                       = 00000001
RT_M_TERM                       = 00000002
RT_V_CASE                       = 00000008
RT_V_TERM                       = 00000009
RT_W_R5                         = 00000000
SCH$LOCKR                         ********    X   02
SCH$LOCKW                         ********    X   02
SCH$UNLOCK                        ********    X   02
SS$_ACCVIO                      = 0000000C
SS$_DUPLNAM                     = 00000094
SS$_IVLOGNAM                    = 00000154
SS$_IVLOGTAB                    = 0000015C
SS$_LNMCREATED                  = 000006B1
SS$_NOLOGNAM                    = 000001BC
SS$_NOLOGTAB                    = 00002294
SS$_NOPRIV                      = 00000024
SS$_NORMAL                      = 00000001
SS$_PARENT_DEL                  = 00002254
SS$_SUPERSEDE                   = 00000631
SS$_TOOMANYLNAM                 = 00000374
```

Psect synopsis

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
ABS	00000000 (0.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE

YF$$LNM 00000788 (1928.) 02 (2.) NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC BYTE

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	35	00:00:00.05	00:00:01.39
Command processing	119	00:00:00.53	00:00:04.44
Pass 1	380	00:00:13.48	00:00:44.17
Symbol table sort	0	00:00:01.66	00:00:05.10
Pass 2	399	00:00:05.69	00:00:17.73
Symbol table output	1	00:00:00.13	00:00:00.70
Psect synopsis output	0	00:00:00.02	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	936	00:00:21.56	00:01:13.55

The working set limit was 2100 pages.
85200 bytes (167 pages) of virtual memory were used to buffer the intermediate code.
There were 60 pages of symbol table space allocated to hold 1052 non-local and 102 local symbols.
2667 source lines were read in Pass 1, producing 18 object records in Pass 2.
23 pages of virtual memory were used to define 22 macros.

Macro library statistics

Macro library name	Macros defined
_255DUA28:[SYS.OBJ]LIB.MLB;1	10
_255DUA28:[SYSLIB]STARLET.MLB;2	9
TOTALS (all libraries)	19

1115 GETS were required to define 19 macros.

There were no errors, warnings or information messages.

```
MACRO/LIS=LIS$:LNMSUB/OBJ=OBJ$:LNMSUB MSRC$:LNMSUB/UPDATE=(ENH$:LNMSUB)+EXECML$/LIB
```

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

IOSUBRAMS
LIS

IPCONTROL
LIS

IOSUBPAGD
LIS

LNMSUB
LIS

IOPERFORM
LIS

IOSUBNPAG
LIS

LINKVEC
LIS

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY